НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

До захисту допущено:

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

Дипломний проект

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Інженерія програмного забезпечення комп'ютерних систем»

спеціальності 121 «Інженерія програмного забезпечення»

на тему: «Клієнтський та серверний додатки: Task Manager та його безпека»

Виконав (-ла):
студент (-ка) 4 курсу, групи ІП-64
Рябінін Андрій Дмитрович _____

Керівник:
асистент,
Ружевський Микита Сергійович _____

Консультант з норма контролю:
професор, доктор технічних наук,
Сімоненко Валерій Павлович _____

Рецензент:
магістр,
Бачинін Іван Вячеславович _____

Засвідчую, що у цій дипломному проекті немає запозичень з праць інших авторів без відповідних посилань.
Студент (-ка) _____

Київ – 2020 року

ВІДОМІСТЬ ДИПЛОМНОГО ПРОЕКТУ

№ з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на	2	
			дипломний проект		
2	А4	ІАЛЦ.467100.001 ТЗ	Технічне завдання	4	
3	А4	ІАЛЦ.467100.002 ПЗ	Пояснювальна записка	79	
4	А4	ІАЛЦ.467100.003 Д1	Поведінкова діаграма:	1	
			діаграма прецедентів		
5	А4	ІАЛЦ.467100.004 Д2	Структурна діаграма:	1	
			діаграма пакетів		
6	А4	ІАЛЦ.467100.005 Д3	Функціональна діаграма:	1	
			діаграма взаємодії		
7	А4	ІАЛЦ.467100.006 Д4	Блок-схема алгоритму	1	
8	А4	ІАЛЦ.467100.007 Д5	Лістинг програми	59	

				ДП 6421. 00.000.00		
	ПІБ	Підп.	Дата			
Розробн.	Рябінін А.Д.			Відомість дипломного проєкту	Аркуш	Аркушів
Керівн.	Ружевський М.С.				1	1
Консульт.	Бачинін І.В.				«КПІ» ім. Ігоря Сікорського Каф. ОТ Гр. ІП-64	
Н. Контр.	Сімоненко В.П.					
Зав.каф.	Стіренко С.Г.					

Пояснювальна записка
до дипломного проекту
на тему: «Клієнтський та серверний додатки: Task Manager та його безпека»

Київ – 2020 року

Національний технічний університет України

«Київський політехнічний інститут імені Ігоря Сікорського»

Факультет інформатики та обчислювальної техніки

Кафедра обчислювальної техніки

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – 121 «Інженерія програмного забезпечення»

Освітньо-професійна програма «Програмне забезпечення комп'ютерних систем»

ЗАТВЕРДЖУЮ

Завідувач кафедри

_____ Сергій СТІРЕНКО

«___»_____20__ р.

ЗАВДАННЯ

на дипломний проект студенту

Рябінін Андрій Дмитрович

1. Тема проекту «Клієнтський та серверний додатки: Task Manager та його безпека», керівник проекту Ружевський Микита Сергійович, асистент, затверджені наказом по університету від « 07 » травня 2020 р. № 1081-с

2. Термін подання студентом проекту: 26.05.2020

3. Вихідні дані до проекту: Технічне завдання

4. Зміст роботи: Загальні положення та опис завдання. Аналіз існуючих рішень, огляд ринку та їх порівняння. Опис архітектури та використаних технологій серверної та клієнтської частин із їх безпекою. Інструкція щодо використання додатку.

5. Перелік ілюстративного матеріалу (із зазначенням плакатів, презентацій тощо):
ІАЛЦ.467100.003 Д1 Поведінкова діаграма: діаграма прецедентів,
ІАЛЦ.467100.004 Д2 Структурна діаграма: діаграма пакетів,
ІАЛЦ.467100.005 Д3 Функціональна діаграма: діаграма взаємодії,
ІАЛЦ.467100.006 Д4 Блок-схема алгоритму.

6. Консультанти розділів проекту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв
Норма контролю	Сімоненко В. П.		

7. Дата видачі завдання 01.09.2019

Календарний план

№ з/п	Назва етапів виконання дипломної проекту	Термін виконання етапів проекту	Примітка
1	Затвердження теми роботи	01.09.2019	
2	Вивчення літератури та предметної області	01.03.2020 - 23.03.2020	
3	Аналіз існуючих рішень	24.03.2020 - 29.03.2020	
4	Вибір технологій і архітектури	30.03.2020 - 04.04.2020	
5	Побудова графічних матеріалів і діаграм	05.04.2020 - 07.04.2020	
6	Писання практичної частини диплома	08.04.2020 - 08.05.2020	
7	Писання теоретичної частини диплома	09.05.2020 - 25.05.2020	
8	Передзахист	26.05.2020	
9	Захист	15.06.2020 - 28.06.2020	

Студент Андрій РЯБІНІН

Керівник Микита РУЖЕВСЬКИЙ

АНОТАЦІЯ

В бакалаврській дипломній роботі було розроблено додаток, головна ідея якого полягає у зручному створені та управлінні користувачем задачами у стилі списку. Архітектура додатку складається із двох частин задля розділення відповідальності: серверна частина та клієнтська частина. Серверна частина розроблена на мові програмування Java за допомогою Spring фреймворку. У свою чергу клієнтська частина створена на базі веб-фреймворку Angular, що базується на мові програмування TypeScript. Ця комунікація між серверною та клієнтською частиною є безпечною і захищеною.

АННОТАЦИЯ

В бакалаврской дипломной работе было разработано приложение, главная идея которого заключается в удобном создании и управлении пользователем задачами в стиле списка. Архитектура приложения состоит из двух частей для разделения ответственности: серверная часть и клиентская часть. Серверная часть разработана на языке программирования Java при помощи Spring фреймворка. В свою очередь клиентская часть создана на базе веб-фреймворка Angular, что базируется на языке программирования Typescript. Эта коммуникация между серверной и клиентской частями является безопасной и защищенной.

ANNOTATION

In this work for a Bachelor's Degree, it was developed the application, the main idea of which lies in convenient way of creating and managing user’s tasks in the style of list. Architecture of application consists of two parts in order to follow the separation of concerns principle: back-end and front-end. Back-end is developed using the programming language called Java with the help of Spring framework. Front-end, in turn, is developed using Angular web framework that is based on the programming language named Typescript. The communication between back-end and front-end is safe and secured.

Технічне завдання

ЗМІСТ

Лист

					ІАЛЦ.467100.001 ТЗ		
Зм.	Арк.	№ документа	Підп.	Дата			
Розробив		Рябінін А.Д.			Клієнтський та серверний додатки: Task Manager та його безпека	Літ.	Аркуш 1 / Аркушів 4
Перевірив		Ружевський М.С.				Т	
Н. контр.		Сімоненко В.П.			Технічне завдання	«КПІ» ім. Ігоря Сікорського Кафедра ОТ, група ІП-64	
Затв.							

1. НАЙМЕНУВАННЯ ТА ОБЛАСТЬ ЗАСТОСУВАННЯ

Поточне технічне завдання поширюється на розробку програмного додатку “Клієнтський та серверний додатки: Task Manager та його безпека”. Область застосування: метод економії часу шляхом коректного і зручного управління задачами і проектами користувачів.

2. ПІДСТАВИ ДЛЯ РОЗРОБКИ

Підставою для виконання бакалаврської дипломної роботи і розробки проекту є завдання на створення додатку з управління задачами та проектами для отримання кваліфікаційно-освітнього рівня «бакалавр», затверджене НТУУ «Київський політехнічний інститут імені Ігоря Сікорського» та отримання навиків з напряму розробки комплексних систем із клієнт-серверною архітектурою.

3. МЕТА І ПРИЗНАЧЕННЯ РОЗРОБКИ

Метою даного програмного додатку є контроль і управління власними задачами та проектами шляхом розбиття задач на логічні блоки і подальше їх опрацювання зручним і ефективним способом як на одинці так і у групі співучасників. Доречне розбиття програмного додатку на клієнтську і серверну частину полегшує розробку і підтримку самої системи інженерами програмного забезпечення.

4. ДЖЕРЕЛА РОЗРОБКИ

В сучасному світі доволі велика кількість аналогів даного додатку, серед яких Trello, Evernote та інші, що стали натхненням і забезпечили новими ідеями щодо створення сервісу. Додатковими джерелами розробки є науково-технічна література з теорії і практики програмування, алгоритмів, шаблонів і публікації в Інтернеті з даних питань.

5. ТЕХНІЧНІ ВИМОГИ

5.1. Вимоги до розробляємого продукту

- Клієнтська частина із комфортним і зрозумілим користувацьким інтерфейсом, що охороняється системою безпеки і потребує авторизації користувача.
- Серверна частина, що займається бізнес логікою та охороняється системою безпеки, яка потребує завчасної авторизації.
- Можливість збереження стану у реляційну базу даних задля повторного відтворення, незалежачи від конкретної реалізації.
- Незалежність використання додатку користувачем, маючи лише доступ до Інтернету через будь-який поширений веб-браузер та незалежність розробки інженером, маючи лише мінімальний набір крос-платформних інструментів.
- Актуальність методів розробки програмного забезпечення та дотримування тону чистої архітектури та чистого коду.

5.2. Вимоги до програмного забезпечення

- Операційна система для настільної системи Microsoft Windows 7/8/8.1/10 або пізніша, Linux Ubuntu 14 або пізніша, macOS. Для телефонів та планшетів Android або iOS.
- Веб-браузер Chrome, Firefox, Opera, Edge, Safari.
- Доступ до Інтернету.

5.3. Вимоги до апаратної частини

- Настільна система на базі процесора Intel або AMD мінімальної потужності, телефон або планшет.
- Оперативна пам’ять (ОЗП) не менше 1 Гбайт.
- Вільне місце на жорсткому диску не менше 1 Гбайт.

6. ЕТАПИ РОЗРОБКИ

	Дата
Вивчення літератури та предметної області	01.03.2020
Аналіз існуючих рішень	24.03.2020
Вибір технологій і архітектури	30.03.2020
Побудова графічних матеріалів і діаграм	05.04.2020
Писання практичної частини диплома	08.04.2020
Писання теоретичної частини диплома	09.05.2020

ЗМІСТ

ВСТУП

Головною метою даної роботи була розробка програмного додатку, ідея якого полягає у створенні, редагуванні, видаленні та управлінні задачами, що створює користувач, у стилі списку.

Здебільшого, люди страждають прокрастинацією, за що їх неможна звинувачувати, так як це є природньо. Люди не вміють досконало структурувати свої задачі, ідеї та цілі. Ці причини стали поштовхом для знаходження зручного способу налаштування та управління своїми задачами, адже людині важливо бачити те, над чим вона працює та спостерігати її прогрес у роботі. Хтось вирішив записувати до блокноту, а хтось малювати діаграми та графіки. Але ми живемо у 21-у столітті і майже у кожній людині є доступ до персонального комп'ютера, ноутбука, телефона або планшета. І майже кожна людина завжди носить при собі хоча б один пристрій, що має доступ до інтернету. Тож чому би не створити такий додаток, який би займався зберіганням задач, що створює людина. Не потрібно купувати олівці, ручки та папір. Не потрібно витрачати час на красиве оформлення тільки задля того, щоб потішити власне "Я".

Для користувача, додаток дозволяє полегшити управління задачами, що мотивує людину на прийняття заходів щодо реалізацій своїх ідей, задач та цілей. Інтуїтивний та зручний інтерфейс клієнтської частини створений таким чином, щоб привернути увагу користувача, потішити його очі та почуття перфекціоніста і мінімаліста. Достатній набір функціоналу одразу стимулює записати всі свої ідеї до списку, а різні списки розділити на категорії, що візуально спрощує і функціонально спрощує використання додатку. Кожен користувач має можливість створити будь-яку кількість дошок, що позначають тип задачі над якою людина буде працювати. Всередині дошки, користувач має можливість додати списки, розділивши задачу на стратегічні кроки. В кожен створений список, людина може додати елемент, що буде мати значення конкретного тактичного кроку в сторону реалізації його стратегії.

Цей елемент, в свою чергу, можна відкрити і записати туди у стилі чату будь-які нотатки та примітки, що було знайдено під час дослідження питання, чого було досягнуто та що залишилось зробити.

Функціонал, що полегшує роботу над задачею та робить її легшою – це можливість додавати до дошки співучасника, який буде мати змогу разом із власником дошки працювати над задачею або ідеєю. Створювати нові елементи, або редагувати існуючі. Переписуватися в чаті, де є можливість обговорювати поточні питання та проводити дискусії.

Для розробника, додаток був створено таким чином, щоб відповідати вимогам сучасного світу. Було використано новітні технології, різні підходи, принципи і архітектурні рішення. Додаток створено саме так, щоб будь-який інший розробник мав можливість підтримувати, удосконалювати та лагодити проект у буд-який час. Таким чином, додаток було розділено на дві частини: серверна частина і клієнтська частина, що розподіляє відповідальність. Таке розподілення дозволяє відокремити розробку на дві команди: перша команда розробників, що займається клієнтською частиною і вони можуть навіть не знати як влаштована серверна частина; друга команда розробників, що займається серверною частиною, які також у свою чергу можуть не знати про те, як влаштована клієнтська частина. Єдине, про що знає команда, що займається клієнтською частиною – це інтерфейс, або API (англ. Application Program Interface), за допомогою якого клієнтська частина спілкується із серверною частиною.

Клієнтська частина розроблена за допомогою такого веб-фреймворку як Angular, що базується на мові програмування Typescript та на таких вже всім відомих технологіях як HTML та CSS.

Серверна частина розроблена на мові Java за допомогою таких новітніх і відомих фреймворків як Spring framework (Spring Core, Spring Data, Spring Security, та інші), Hibernate, Gradle. Використовується така база даних як MySQL, але це не є обов’язковою умовою і при бажані можливо доволі легко мігрувати на іншу базу даних.

Ці дві частини додатку мають чітку структуру та розділені на певні архітектурні шари абстракцій для ще більшого розділення відповідальності, що полегшує будь-яку розробку та модифікацію програмного додатку розробником.

Якщо ж питання зайде про фінансову складову, то вигоду із такого проекту можливо отримувати за допомогою платних преміум акаунтів, що можуть мати розширений список функціоналу з будь-якою кількістю платних планів із різним функціоналом, що стає доступним при придбанні певного плану. Такий платний контент не є обов’язковим для користувача, який має можливість без будь-яких проблем використовувати додаток із безкоштовним мінімальним набором функціоналу. Також можливо проводити інтеграцію та співпрацю з іншими проектами.

Варіант із рекламою не береться до розгляду, адже реклама збиває з пантелику людей та відбиває велику кількість потенційних користувачів, що в свою чергу впливає на популярність та успіх додатку та проекту у цілому.

РОЗДІЛ 1. АНАЛІЗ ІСНУЮЧИХ АНАЛОГІВ РІШЕНЬ

1.1. Опис завдання

Завдання цієї роботи полягає в створені програмного додатку, що складається із двох частин задля розділення відповідальності: серверна частина та клієнтська частина. Серверна частина представляє собою мозок усього програмного додатку, так як вся логіка, що присутня, знаходиться на серверній частині, та дані, які потребують довготривалого зберігання також проходять серверну частину та зберігаються у базі даних. Клієнтська частина ж являє собою обличчя програмного додатку. За допомогою цієї частини користувач виконує дії щодо користування додатку та спілкується із серверною частиною через обгортку, що ще має таку назву як користувацький інтерфейс (англ. UI – user interface).

Головна ідея програмного додатку полягає у налаштуванні роботи або задачі користувачем шляхом розбиття самої задачі на стратегічні та тактичні шаги. Таким чином, кожен користувач має можливість створити будь-яку кількість так званих дошок, що представляють собою головну робочу область. Дошка, в свою чергу, розбивається на колонки. А колонки розбиваються на рядки. Кожен рядок вміщає в собі функціонал подібний чату, що дозволяє нотувати інформацію та записувати потрібні користувачеві дані. До кожної дошки її власник має можливість додати співучасника. С такою співпрацею власника дошки і будь-якою кількості співучасників досягається висока ефективність роботи над будь-якого роду задачами та досягається гнучка системи контролю цих задач.

Варто підмітити, що така система є безпечною і захищеною від зловмисників. Будь-яка інформація, що додається користувачем на клієнтський частині, проходить серверну частину і зберігається у базі даних є абсолютно конфіденціальною і ніхто крім власника не має можливості переглянути або ж заволодіти цією інформацією. Для досягнення такої мети

використовуються новітні та ефективні технології, системи і рішення задля забезпечення комфортного та надійного використання програмного додатку. Почуття та відносини користувача є важливим аспектом, але також важливим є і фактор довіри своїх даних та інформації користувача розробнику системи

1.2. Аналіз існуючих рішень

Насправді існує велика кількість можливих подібних систем по контролю задачами. Адже людина не в перший раз задається питанням ефективного розподілення своїх задач та подальшої зручної роботи над ними в будь-який час. Ефективність розподілення та налаштування задач також впливає на ефективність їх виконання та якість кінцевого результату.

1.2.1. Сервіс контролю задачами “Todoist”

Todoist – це веб-сервіс та набір функціоналу, що забезпечує контроль над задачами. Задачі мають можливість зберігати нотатки із файлами будь-якого типу. Задачі можливо поміщати у проекти, сортувати за фільтром, редагувати та експортувати задля збереження ззовні веб-сервісу.

Todoist підтримує на сьогодні тринадцять комп’ютерних та мобільних систем, серед яких: Microsoft Windows, Apple Mac OS X, iOS, Android, Firefox, Google Chrome, Gmail, Microsoft Outlook, Postbox та Thunderbird. Також він забезпечує можливість синхронізації серед різних пристроїв та резервне зберігання даних. На сьогодні ж не має підтримки такої популярної операційної системи як Linux.

Задля надання мотивації, платформа використовує систему “карми”, що допомагає аналізувати та візуалізувати продуктивність та ефективність користувача та показувати статистику по виконаним задачам у барвистих та зручних графіках. Дотримуючись обраного вами шляху, вказуючи число задач, які ви плануєте виконати протягом дня або неділі, ви отримуєте певну кількість карми. Для того, щоб ви могли поділитися своїм прогресом, існує

функціонал, що дозволяє розказати про свої досяги за допомогою таких соціальних мереж як Facebook та Twitter. Всього існує п’ять рівнів, що залежать від кількості набраних користувачем умовних балів: Новачок, Любитель, Експерт, Майстер і Гуру.

У веб-сервіса Todoist існує можливість преміум та бізнес акаунтів (див. таблицю 1.1). [19]

Таблиця 1.1

Порівняння тарифів

Функціональність	Безкоштовний	Преміум	Бізнес
Кількість проектів	80	300	500
Співробітників в проекті	5	25	50
Повторювані терміни виконання	+	+	+
Пріоритети задач	+	+	+
Інтеграції	+	+	+
Нагадування	-	+	+
Коментарі і загрузка файлів	-	+	+
Мітки	-	+	+
Фільтри	-	+	+
Тренди продуктивності	-	+	+
Преміум теми	-	+	+
Автоматичне резервне копіювання	-	+	+
Додавання задач по Email	-	+	+
Сінхронізування із календарем	-	+	+
Шаблон проектів	-	+	+
Журнал дій	-	+	+
Пріоритетна підтримка	-	+	+
Загальні вхідні для команди	-	-	+

Таблиця 1.1 (закінчення)

Функціональність	Безкоштовний	Преміум	Бізнес
Ролі адміна і учасника	-	-	+
Рахунки команди	-	-	+

Робота із Todoist починається із головного вікна, що виглядає лаконічно та мінімалістично (Рис. 1.1).

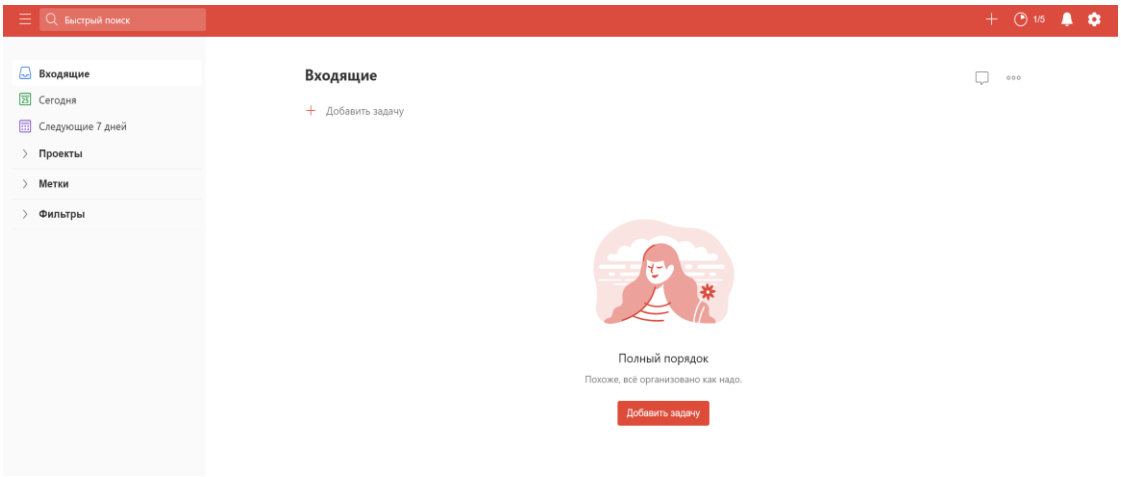


Рис. 1.1 Головне вікно

Із головного вікна користувач має можливість одразу створити задачу або ж подивитися свої вхідні задачі, задачі на сьогодні або наступну неділю. Також є змога відкрити та подивитися усі свої проекти, мітки та фільтри на бічній планці (Рис. 1.2).

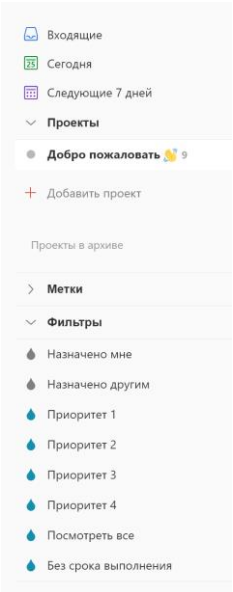


Рис. 1.2 Бічна планка

Додавання новою задачі виглядає як вікно де користувач має змогу ввести назву, визначити термін виконання, додати задачу до проекту, повісити мітку або задати пріоритет, додати нагадування або ж додати підзадачу та батьківську задачу (Рис. 1.3).

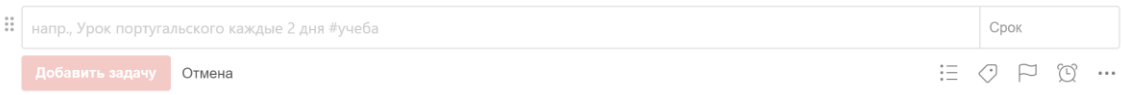


Рис. 1.3 Вікно додання задачі

Після додавання новою задачі, користувач може видалити, редагувати, додавати коментарі, переміщати, видаляти та інший набір функціоналу, що був доступний під час створення задачі.

Поміж основного функціоналу управління задачами, є можливість відкрити налаштування профілю, зміні фон та тему, застосувати шаблони, роздрукувати задачі, звернутися до служби підтримки, почитати тематичний блог і новини. Є можливість пошуку задач та перегляд своєї продуктивності. Якщо є якісь повідомлення, то їх можна переглянути в доступному меню.

1.2.2. Сервіс запису нотаток “Google Keep”

Google Keep – це сервіс для запису нотаток, що був розроблений компанією Google. Він є абсолютно безкоштовний та немає платних акаунтів або планів із додатковим функціоналом. Цей сервіс доступний через веб-сайт, а також має мобільні додатки для таких операційних систем як Android та iOS. Google Keep пропонує різноманітні інструменти для нотування: текст, списки, картинки і аудіо записи. Користувач має можливість встановити нагадування, витягнути текст із картинки використовуючи технологію оптичного розпізнавання символів, а аудіо записи можливо перетворити у текст. Присутній функціонал перетворення тексту нотаток у список, або так називаємий чекліст. Можлива інтеграція із Google Docs, що дає можливість в один клік скопіювати увесь текст нотатки до Google Docs документу. А щоб

класифікувати створені нотатки, можливо додати будь-які мітки використовуючи перелік створених користувачем міток.

Інтерфейс дозволяє переглядати нотатки у режимі одного стовпчика або у режимі багатьох стовпчиків. А самі нотатки можуть бути кольоровими, мати на собі мітки задля полегшення їх управління, та також їх можливо закріплювати, щоб не загубити найважливіші серед них. Keep має функціонал співпраці, що дозволяє користувачам об'єднуватись разом і вирішувати питання та задачі координуючи один одного. [9]

Робота із Google Keep починається із головного вікна, що у веб версії додатка виглядає наступним чином (Рис. 1.4).

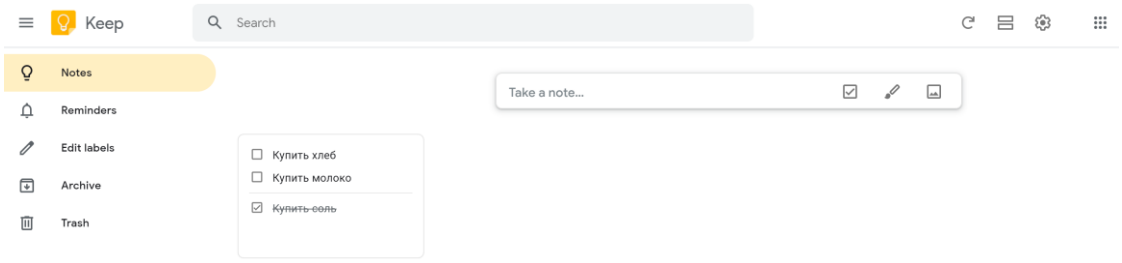


Рис. 1.4 Головне вікно

Із даного головного вікна можливо одразу перегляну створені користувачем нотатки та взаємодіяти із ними редагуючи або видаляючи. Також присутнє поле для пошуку, що полегшує навігацію по доданим нотаткам, якщо так вийшло, що їх велика кількість.

Додання нового нотатку виконується в один клік нажимаючи на поле, що знаходиться зверху над усіма вже створеними нотатками. А вікно для додання виглядає стилістично та функціонально комфортним (Рис. 1.5).

Рис. 1.5 Вікно додання нотатку

Додаючи нову нотатку, користувач має деякий набір інструментів і функціоналу, що робить користування більш ефективним і продуктивним, поліпшуючи управління та контроль над задачами і мотивуючи людину до роботи. Серед інструментів присутній наступний функціонал:

- задати назву і звичайно сам текст;
- виставляти нагадування на певний час, або навіть певне місце, приходячи до якого, людина отримає повідомлення про нотатку;
- додання співучасника для командної роботи над задачею або питанням;
- виставлення кольору;
- додання картинки майже будь-якого розширення
- додання мітки;
- додання малюнку, створеного користувачем одразу в додатку;
- створення листу із чек-боксами;
- прикріплення нотатку до важливих.

Веб-сайт додатку Google Keep також вміщає в собі бокову планку із деякою кількість сторінок, які користувач має можливість відвідати (Рис. 1.6).

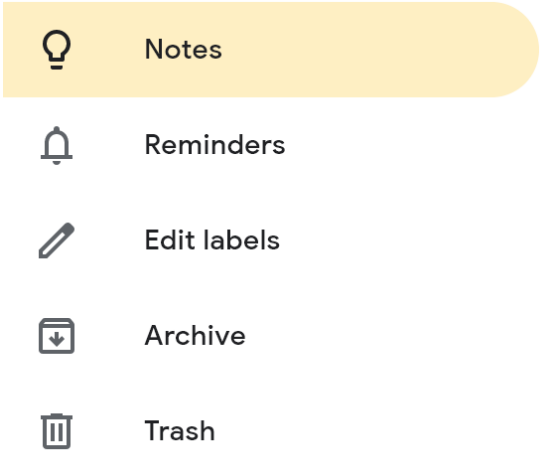


Рис. 1.6 Бічна планка

Ця бічні планка включає такі сторінки як сторінка із доданими нотатками, де користувач має можливість додавати, редагувати, видаляти та модифікувати свої нотатки; сторінка із нагадуванням, що показує нотатки, до

яких було прикріплено функціонал нагадування; сторінка із створеними мітками, де можна додавати нові або видаляти існуючі мітки, які кріпляться до нотаток; сторінка із архівними нотатками, куди потрапляють нотатки, що були примусово переміщені користувачем до архіву; та сторінка із видаленими нотатками, що потрапляють до смітника одразу після їх видалення та зберігається там в протягом семи днів або поки користувач вручну не видалить їх звідти.

Посеред іншого, присутні налаштування, що дозволяють кастомізувати Google Keep. Користувач може змінювати тему із денної на нічну та навпаки, якщо він цього потребує. Присутня система допомоги та система залишення відгуку. Можливо вибрати режим перегляду нотаток, а саме: перегляд у режимі списку та перегляд у режимі сітки. Існує система гарячих клавіш, що доволі непогано прискорює управління нотатками, тобто користувачеві не потрібно завжди тримати комп'ютерну миш у руці, бо натискаючи певні комбінації на клавіатуру, він має можливість: переходити до наступної або попереднього нотатку, додавати нову нотатку або список, використовувати пошук, виділяти усі нотатки, архівувати, видаляти, прикріпляти та інший функціонал, що присутній у додатку.

Останнє, але не менш важливе, це те, що даний додаток є продуктом компанії Google, що додає такий функціонал як інтеграція із іншими продуктами цієї компанії, а саме: можливість конвертації і перенесення нотаток до Google Docs документу, можливість перегляду нотаток із Google Keep одразу у Google Docs документі або навіть у Gmail пошті.

1.2.3. Додаток контролю задач "Evernote"

Evernote – це додаток для запису нотаток, організування та контролю задач. Він дає можливість створювати примітки використовуючи текст, малюнки, фотографії або збережений веб контент. Нотатки зберігаються в зошитах і їх можна позначати тегами, коментувати, редагувати, шукати, прикріпляти додатки та експортувати. Evernote є кросплатформеним

додатком, що дає можливість використовувати його на Android, iOS, macOS, Microsoft Windows та з усіма відомими веб браузерами.

При першому відвідуванні додатку, користувача зустрічає навчальний посібник, що допомагає розібратися із функціоналом додатку та зрозуміти основні моменти щодо комфортного користування ним.

Найсильніша сторона Evernote – це зберігання всього підряд. Чим більш матеріалу користувач зберігає в блокнотах, тим ефективніше він буде працювати. Протягом дня людина зустрічає безліч цікавих та корисних веб ресурсів, статей, заміток. І в той самий момент коли цей корисний ресурс знаходиться, зазвичай не вистачає часу для того, щоб прочитати та обробити інформацію в повну силу. [8]

Робочий простір в Evernote поділено на декілька областей наступним чином (Рис. 1.7):

- Деревоподібна структура ваших блокнотів, міток та ярликів
- Перелік заміток в обраному блокноті
- Тіло вибраної замітки

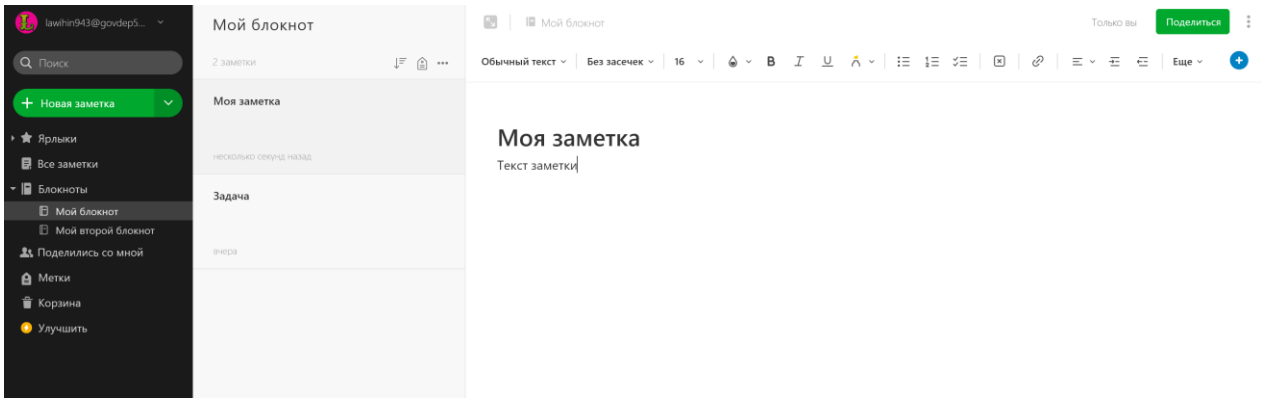


Рис. 1.7 Робочий простір

Загалом, додаток дозволяє створювати, видаляти та редагувати замітки. Замітки самі по собі зберігаються в так званих блокнотах, які користувач може також як створювати так редагувати і видаляти. Для комфортного контролю замітками, присутній функціонал міток та ярликів. Останні дають можливість миттєвого переходу до часто використовуваних заміток та блокнотів, якщо

вже присутня їх велика кількість і знаходження курсором потребує деякого часу.

Додання нової замітки виконується в один клік, після чого відкривається вікно із редактором заміток, де користувачу відкривається величезний набір інструментів для роботи із текстом: розмір тексту та його стиль, колір шрифту, курсив, підкреслення, виділення, додання маркірованого та нумерованого списку, додання списку у стилі чеклісту, вставка посилання, вирівнювання, додання отступів, перекреслення тексту та інший набір функціоналу (Рис. 1.8).

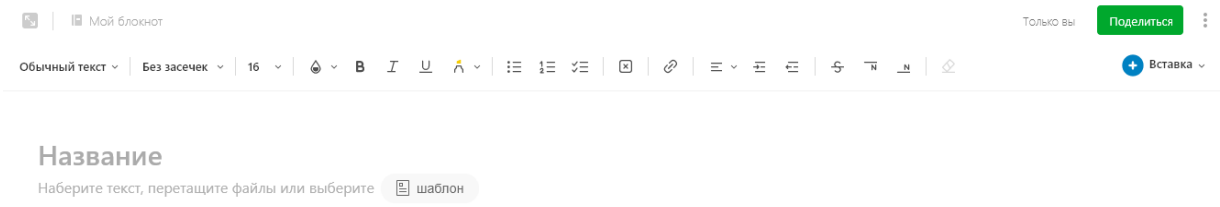


Рис. 1.8 Створення нової замітки

Замість створення замітки з нуля, є можливість обрати існуючи стилі із шаблонів, що пришвидшує роботу над задачею, наприклад шаблон щоденника, бюджетного проекту або матриці Ейзенхауера. Таким чином людина економить час на оформленні та одразу приступає до опису задачі та її виконанню.

Для більшої безпеки користувач додатку на операційних системах Windows та Mac має можливість зашифрувати будь-який текст в нотатках, щоб додати додатковий рівень захисту для особистої інформації, наприклад інформації про акаунт, планів поїздки і особистих листів. Є можливість зашифрувати текстовий вміст замітки, але не замітку або блокнот цілком.

Одною із приємних особливостей додатку є можливість додання геотегів до заміток. Ці самі геотеги або ж координати користувач може використати в картах Google Maps.

Work Chat – одна із чергових особливостей додатку, що дозволяє зручно обмінюватися замітка в Evernote та підтримувати сумісну із колегами роботу.

Є можливість ділитися певними замітками із учасниками чату і видавати підходящі права, наприклад права на можливість редагувати, видаляти або тільки переглядати замітку.

Синхронізація ж допомагає користувачу знаходити і додавати замітки на одному пристрої і використовувати ці додані замітки на будь-якому іншому пристрої що має підтримку від додатку. Таким чином, переглядаючи статтю на телефоні і дізнавшись про те, що залишилось мало вільного часу, людина має можливість зберегти цю статтю і як тільки з’явиться вільна година, перечитати та обробити її на комп’ютері у повному комфорті.

Розширення для більшості браузерів Web Clipper дозволяє зберігати одразу усю веб-сторінку або тільки деякі її частини без зайвої реклами або заголовків. Присутня можливість зробити знімок екрану, додати текст, форми, коментарі, додавати теги, зауваження. Якщо потрібно, можна відредагувати заголовок сторінки. Після цього, знімок зберігається в будь-який обраний блокнот. Цей функціонал дозволяє швидко додати до нотаток знайдену інформацію, і що головніше, зберегти її таким чином, щоб повернувшись через деякий час до замітки, зрозуміти навіщо і для чого веб-сторінка або її знімок був була збережена. (Рис. 1.9)

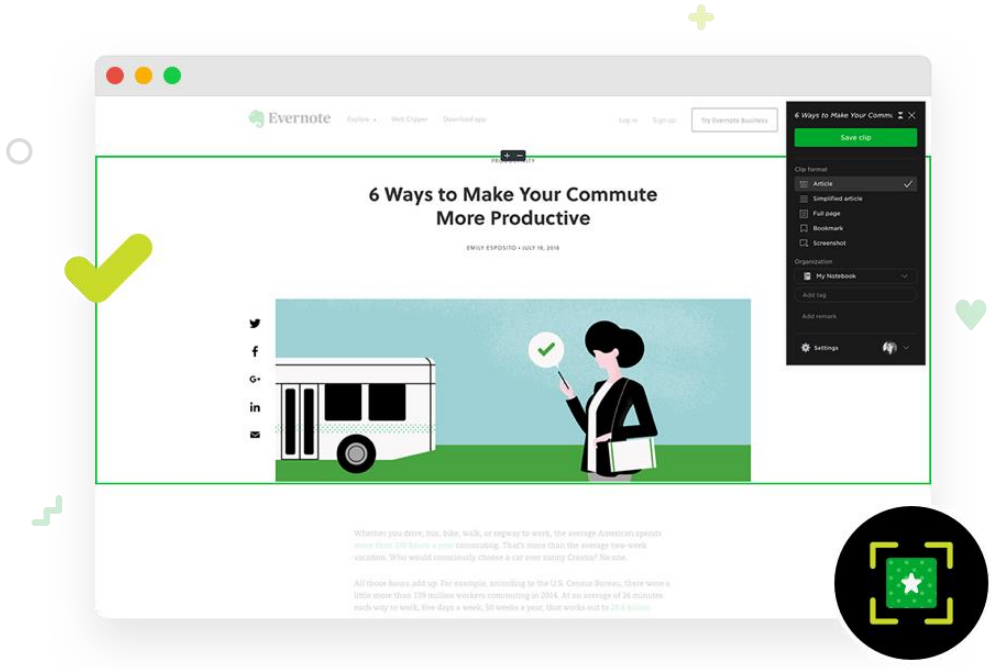


Рис. 1.9 Робота із розширенням [**Ошибка! Источник ссылки не найден.**]

Сервіс Evernote має декілька планів, що відкривають різний функціонал в залежності від самого плану, серед них: безкоштовний, преміум та бізнес (таблиця 1.2).

Таблиця 1.2

Порівняння планів

Функціональність	Безкоштовний	Преміум	Бізнес
Синхронізація на різних пристроях	До двох	Необмежено	Необмежено
Щомісячні завантаження	60 MB	10 GB	24 GB+
Максимальний розмір замітки	25 MB	200 MB	200 MB
Підтримка клієнтів	Форум	Пошта/чат	Телефон
Доступ до ноутбуків офлайн	-	+	+
Додавати замітки через Email	-	+	+
Пошук в Office Docs і PDF	-	+	+
Представлення заміток в 1 клік	-	+	+
Анотування PDF-файлів	-	+	+
Керування командними даними та доступом	-	-	+
Співпраця у спільних просторах	-	-	+

І це є лише поверхневим представленням доступу до функціоналу в залежності від обраного користувачем плану. В цілому, список доступного функціоналу дуже різниться, а головне те, що загалом для щоденних цілей та більшості задач вистачає безкоштовного плану.

Підводячи підсумок, Evernote - це сервіс, який представляє собою просунутий робочий простір для збільшення продуктивності будь-якого користувача. Хтось використовує сервіс тільки для ведення своїх особистих справ, проектів або ж просто для нотування, а хтось занурює в систему всі

записи про свій бізнес використовуючи додатковий, доволі різноманітний, функціонал.

1.2.4. Система управління проектами “Trello”

Trello – це система управління проектами у стилі списку. Функціональність дозволяє створювати дошки для задач із колонками всередині і пересувати задачі між цими колонками. Сервіс доступний на таких операційних системах як macOS, Windows, Android, iOS та у будь-якому сучасному веб браузері.

Стиль, який використовує Trello, називається Kanban, а якщо точніше, то Kanban дошки. Дошка Kanban – це один із інструментів за допомогою якого можна реалізувати Kanban для управління роботою на проектам як на особистому рівні, так і на рівні організації. Дошки візуально відображають роботу на різних етапах процесу, використовуючи картки для представлення задач та стовпці для представлення кожної стадії робочого процесу. Зазвичай картки переміщуються злів направо, задля того, щоб показати прогрес і допомогти координувати команди, що працюють над цими задачами. У найпростішому представленні зазвичай присутні наступні стовпці, що відображають етапи процесу роботи:

- “To do”, відображає етап на якому задачі готові до виконання
- “Doing”, відображає етап на якому задачі знаходиться в процесі виконання
- “Done”, відображає етап до якого потрапляють задачі після їх успішного виконання

Звичайно можуть бути створені і більш складні системи дошок Kanban, що дозволяє кастомізувати процес роботи і візуалізувати етапи виконання у багатьох стилях задовольняючи більшість потреб.

В першу чергу, додаток Trello зустрічає нового користувача із невеликого навчального посібника, щоб занурити його у простий і початковий функціонал сервісу і облегшити використання самого сервісу та роботу із ним. [20]

Зазвичай, система зустрічає користувача із начального екрану, що містить усі створенні дошки, дозволяючи переходити всередину цих дошок і одразу починати роботу над задачами або проектами (Рис. 1.10).

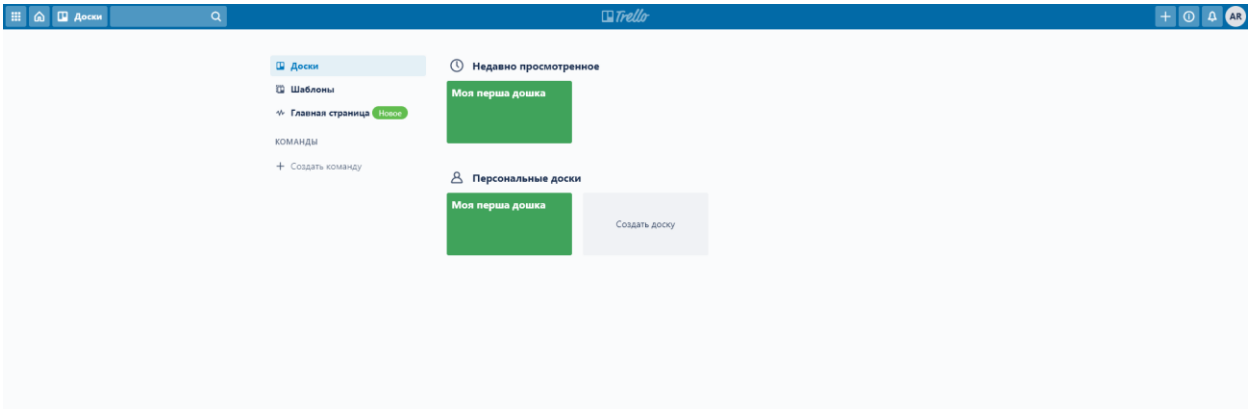


Рис. 1.10 Начальний екран

Із начального екрану користувач має можливість створити нову дошку або перегляну уже існуючи, продивитися шаблони, переглянути головну сторінку або ж створити команду.

Створення нової дошки не потребує детального опису або аналізу, адже усе, що потрібно зробити, щоб створити нову дошку – це один клік комп'ютерної миші із подальшим введенням назви самої дошки.

Після створення нової дошки або переходу до вже існуючи дошки, користувач потрапляє на сторінку, де і знаходиться головний функціонал сервісу по управлінню задачами (Рис. 1.11).

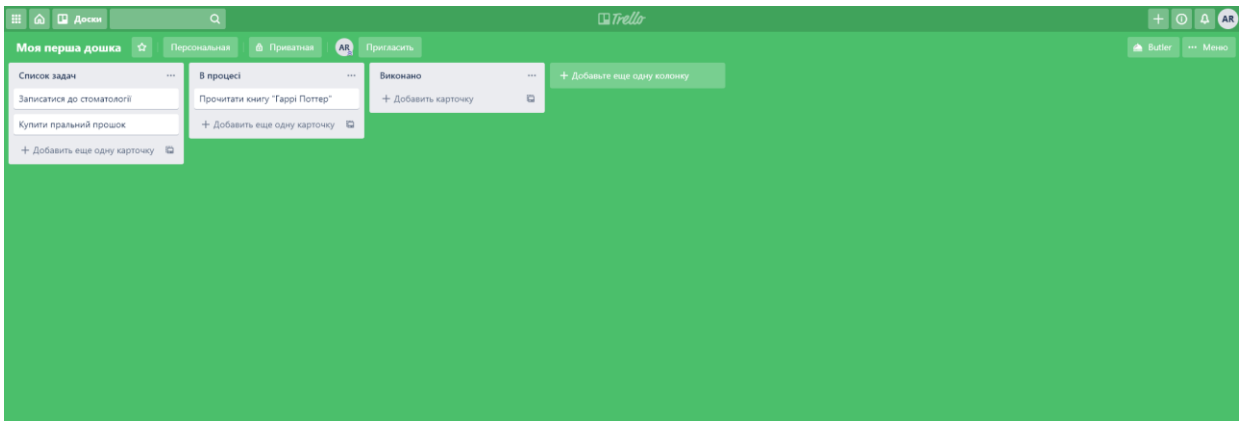


Рис. 1.11 Сторінка дошки

Потрапляючи до сторінки дошки, користувач має можливість створювати нові колонки, додавати до колонки нові задачі та переміщати як створенні колонки між собою так і створенні задачі.

Створення самої колонки або задачі виконується в одне натискання миші, після чого користувач потрапляє до вікні із набором назви. Таким чином, створення нового елементу займає лічені секунди і не потребує довготривалих зусиль.

Проте після створення нової задачі, користувач має можливість відвідати вікно самої задачі (Рис. 1.12).

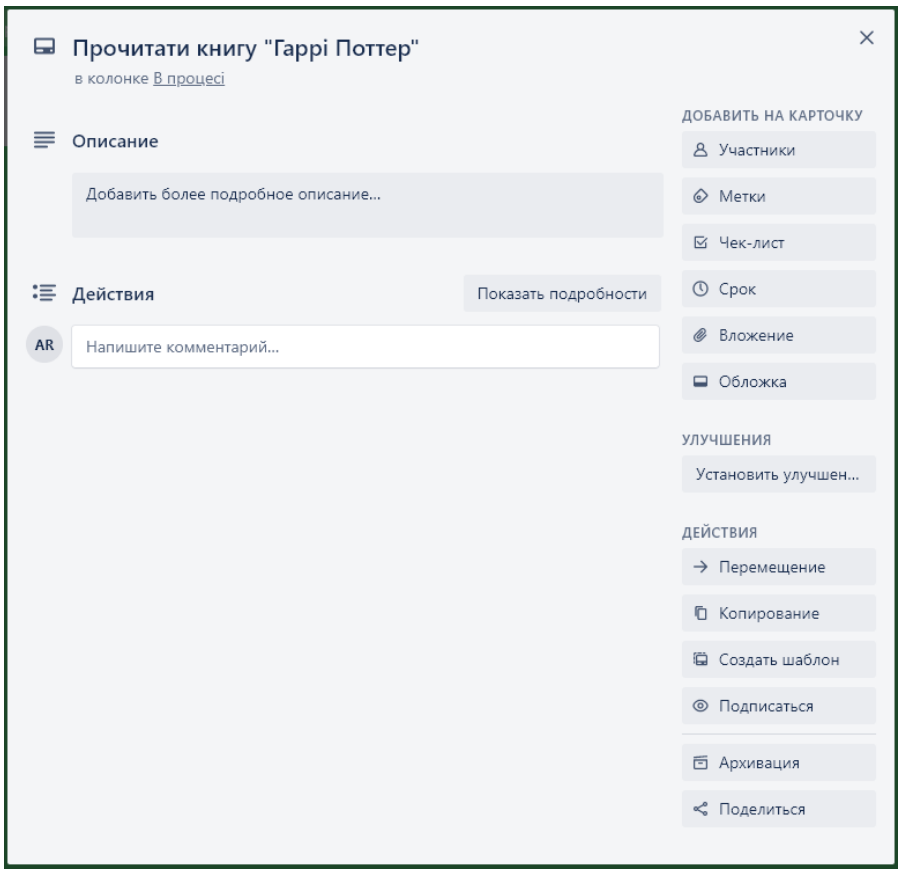


Рис. 1.12 Вікно задачі

Перед користувачем відкривається доволі великий набір інструментів та функціоналу, що надається сервісом Trello. В цьому вікні є можливість змінити назву задачі, додати опис та коментар. Як опис, так і коментар можуть містити звичайний текст, посилання на веб-сайт або іншу задачу, прикріплені картинки або відеозаписи та згадування учасника, якщо над дошкою працює команда мінімум із двох людей.

Із вікна можливо додати до задачі учасники, повісити мітки, додати чекліст, вказати терміни виконання задачі, додати вкладення або змінити обкладинку задачі. Також прямо із вікна задачу можливо перемістити до іншої колонки або скопіювати її. Функція підписки на задачу дозволяє не пропустити будь-які зміни зв’язані із самою задачею – переміщення між колонками, редагування опису або додання коментарю одразу оповістить людину, що підписалася. Якщо людина не підписана на задачу, але потрібно її сповістити, то можливість згадування користувача під час створення коментарю виконає таку потребу. З цього ж вікна задачею можливо

поділитися із іншим користувачем. А якщо задачу по будь-яким причинам потрібно видалити, але є можливість, що інформація про неї може знадобитися, то задачу можливо архівувати і повернутися до неї за такою потреби.

Варто помітити, що додаток має багатий функціонал інтеграції із різними іншими сервісами та так званий функціонал поліпшень, що дозволяє полегшити більшість роботи або навіть автомізувати її. Список поліпшень великий і включає у себе безліч функцій, що дають можливість кастомізувати і модифікувати робочий простір сервісу, дошки, колонки та задачі. Серед типів поліпшень є наступні:

- Аналіз
- Автоматизація
- Комунікація і взаємодія
- Інструменти розробника
- Управління файлами
- Продукт і дизайн
- Продаж і підтримка

Але навіть цей список є не повним і не відкриває усю силу функціоналу поліпшень, так як ця функція є доволі гнучкою системою, що пропонує безліч поліпшень та удосконалень (Рис. 1.13).

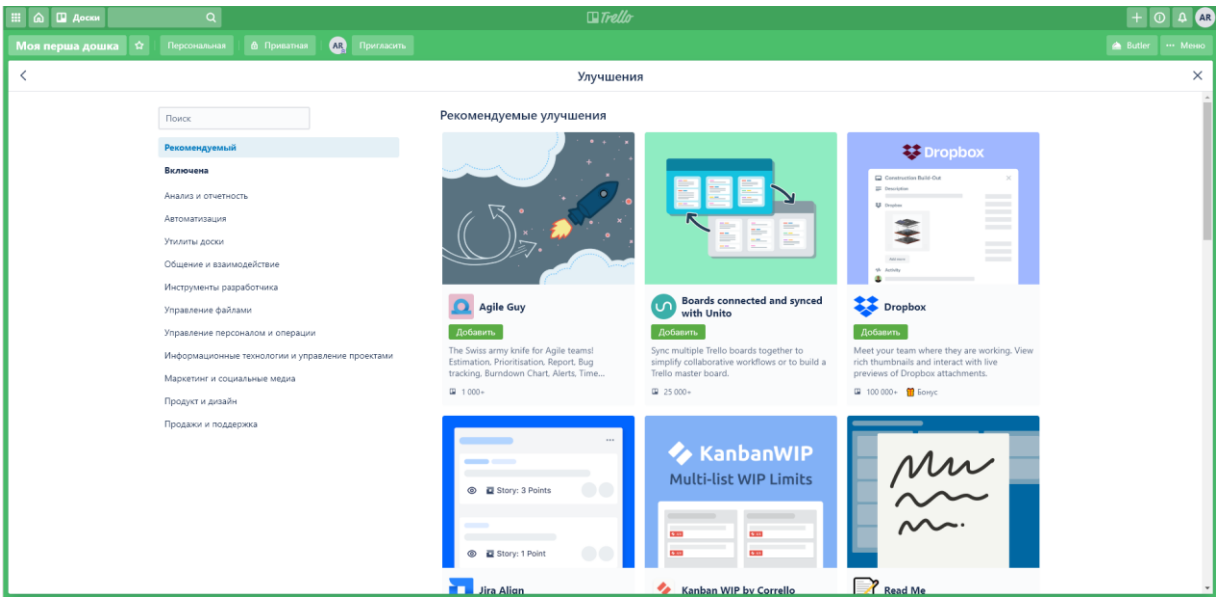


Рис. 1.13 Вікно поліпшень

Сервіс Trello має також і додатковий різноманітний функціонал, що відкривається в залежності від присутнього у користувача плану. В цілому існує три плани: безкоштовний, бізнес, корпоративний (таблиця 2.3).

Таблиця 1.3

Порівняння планів

Функціональність	Безкоштовний	Бізнес	Корпоративний
Кількість дошок	Необмежено	Необмежено	Необмежено
Кількість карток	Необмежено	Необмежено	Необмежено
Кількість списків	Необмежено	Необмежено	Необмежено
Максимальний розмір вкладеного файлу	10 MB	250 MB	250 MB
Користувацькі фони й наліпки	-	+	+
Оглядачі	-	+	+
Пріоритетна підтримка	-	+	+
Календар	-	+	+
Голосування	-	+	+
Інтеграція із іншими сервісами	-	+	+
Адміністрування підсилювачів	-	-	+

Таблиця 1.3 (закінчення)

Функціональність	Безкоштовний	Преміум	Бізнес
Обмеження вкладень	-	-	+
Дозволи організації	-	-	+
Дошки для організації	-	-	+
Управління публічними дошками	-	-	+

Таблиця є лише коротким описом планів із набором функціоналу, що присутні у сервісі. Хоча набір інструментів відрізняється в залежності від плану, але важливо підмітити, що для комфортної роботи над більшістю щоденних задач достатньо навіть безкоштовного плану.

Таким чином, сервіс Trello – є гнучким у багатьох питаннях стосовно поліпшення та прискорення роботи над задачами і проектами комфортним та приємним шляхом. Додаток пропонує багатий набір функцій і інструментів, включаючи співпрацю у команді із іншими користувачами.

ВИСНОВКИ ДО РОЗДІЛУ 1

Можна побачити, що насправді існує доволі велика кількість сервісів та додатків, що займаюся управлінням та контролем задачами та проектами. Усі вони дають можливість створювати, редагувати та видаляти елементи. Для більш ефективної роботи, кожен із переглянутих сервіс має можливість співпраці із іншими користувачами, що дозволяє розподілити робочі сили порівну між співучасниками задля прискорення процесу та швидкого отримання адекватного результату. Весь цей стандартний функціонал контролю задачами також присутній і в додатку проекту дипломної роботи і надає його таким чином, що схожий на сервіс Trello, у стилі списку, тобто створюючи дошки, колонки та задачі із можливістю їх подальшого переміщення.

При створенні або редагуванні задачі, кожен із описаних сервісів має доволі гнучку можливість стилізації тексту, прикріплення файлів, зображень, додання міток, задля зручного подальшого пошуку і нагадувань, задля того, щоб не пропустити важливу подію, зустріч чи загалом розгляд важливого питання. Дипломний проект має доволі скромний функціонал стилізації та модифікації тексту. Тобто єдина присутня можливість роботи із тексту – це його додання. Можливість подальшої стилізації, модифікації або прикріплення будь-яких файлів, міток й нагадувань – на цей час в арсеналі функціональності відсутня. Хто може побачити це як мінус, а хто навпаки як плюс, аргументуючи це максимальним фокусом на праці і на роботі над задачами та проектами, уникаючи марну трату часу на роботу із гнучким й багатостороннім оформлення самих задач. Багата функціональність не всіх і не завжди мотивує до роботи, адже маючи велику кількість функцій, потрібно витратити час на навчання використання цих самих функцій й оволодіти розумінням їх потреби.

Деякі додатки, як наприклад Trello, мають вбудовану систему комунікації із співучасниками по команді, що прибирає бар’єр в спілкуванні і дискусії

щодо завдання або задачі. Такий інструмент надає можливість не залишаючи додаток обговорити поточне питання й дійти до спільного знаменнику із іншими учасниками. Система, що створена для дипломної роботи має в своєму арсеналі функціонал вбудованої системи комунікації, що ще можна назвати як чат. Цей функціонал представляє собою відправку та отримання повідомлення для співучасників в межах конкретної задачі, що поліпшує якість результату виконання.

Більшість із сервісів підтримують такі популярні й сучасні операційні системи як Windows, macOS, Android, iOS й велику кількість сьогоденних веб-браузерів, серед яких Google Chrome, Mozilla Firefox, Opera та інші. Програмний додаток бакалаврської роботи підтримує на сьогодні лише сучасні веб-браузери із відсутністю підтримки як комп'ютерних операційних систем так і мобільних.

Можна підмітити, що кожен додаток (окрім Google Keep) має інструмент для обрання користувачем різних планів, що надають різний й розширений набір функціоналу в залежності від обраного плану. Зазвичай присутні три види планів: безкоштовний, покращений й просунутий. Таким чином перед користувачем відкривається багатий світ функцій й можливостей, що дають спроможність кастомізувати свій робочий простір й налагодити процес гнучким способом. Кожен із планів, окрім першого, потребує внесення коштів щомісяця або щороку. Тобто однією із моделей заробітку сервісів є підписка користувачів на платні плани, а в замін на підписку, користувач отримує додатковий, різноманітний набір інструментів, що поліпшують й прискорюють процес його праці. Доречно сказати, що насамперед, кожен із сервісів надає можливість працювати із безкоштовним планом, чого вистачає на комфорту й продуктивну працю над більшістю тривіальних проектів або щоденних задач. Дипломна робота ж не має такого функціоналу надання різних планів користувачам. Будь-яка людина, що використовує додаток, має один єдиний план, що є безкоштовний і вміщає у собі усі доступні інструменти для роботи із задачами й проектами. Тобто, сервіс втрачає

можливість монетизації проекту шляхом підписок на різні плани, але в той же час, надає користувачам єдиний і повний набір інструментів й функцій, що є доступними будь-якому користувачу у будь-який час.

Загалом, переглянуті сервісі мають як недоліки так і переваги, порівнюючи із даною дипломною бакалаврською роботою. Важливо признати, що переваг в плані функціональності й інструментарію більше, але як вже було підмічено, для когось багатий набір інструментів це перевага, а для когось недолік, адже це в деякій мірі сприяє розфокусуванню й втраті концентрації людини.

РОЗДІЛ 2. ОПИС АРХІТЕКТУРИ ТА ВИКОРИСТАНИХ ТЕХНОЛОГІЙ РОЗРОБКИ

2.1. Архітектура програмного забезпечення

2.1.1. Розподілення відповідальності

В інженерії програмного забезпечення існує такий термін як розподілення відповідальностей (англ. separation of concerns). Відповідальності розподіляються на клієнтську частину, що задовольняє презентаційний рівень, і серверну частину, що задовольняє рівень доступу до даних.

В архітектурі програмного забезпечення можлива багата кількість шарів між апаратними засобами (англ. hardware) й кінцевим користувачем. Клієнтська частина – це шар абстракції, що спрощує комунікацію із користувачем шляхом надання зручного і зрозумілого інтерфейсу, в той час як серверна частина займається зберіганням та обробкою даних та бізнес-логіки, що приховується від користувача за клієнтською частиною. Якщо спростити визначення, то клієнтська частина це те, як додаток виглядає, а серверна частина це те, як додаток працює.

Таке розподілення спрощує роботу над сервісом і дає можливість, незалежно один від одного, змінювати імплементацію частин й також дозволяє розділити саму роботу на дві частини, виділивши абсолютно незалежні команди, які можуть навіть не знати про те, як реалізована інша із частин. [4, 5]

2.1.2. Класифікація мов програмування

2.1.2.1. Стилі програмування

Загалом виділяють два основні стилі мов програмування (в цілому їх існує достатня велика кількість, проте найбільше застосування набрали два

основні стилі): імперативний підхід програмування й декларативний підхід програмування.

Найбанальнішим описом різниці цих підходів виглядає таким чином, що імперативне програмування – це опис того, яким чином ти виконуєш дію, а декларативне – це опис того, що саме ти робиш і який потрібен результат. Можна додати наступну метафору для більшого розуміння. Задавши питання “Я знаходжусь біля банку. Як мені дібратися до тебе?”, можливо отримати два типи відповіді:

- Імперативний: “Тобі потрібно стати лицем до сторони банку, повернутися направо й пройти до кінця дороги. Перейшовши пішохідний перехід, поверни наліво й пройди до ювелірного магазину. Я буду знаходитись під вивіскою самого магазину.”
- Декларативний: “Я знаходжусь по адресу: проспект Миру 10, корпус 14.”

Важливо також зрозуміти той факт, що багато декларативних підходів мають деякий шар імперативних абстракцій. Так, повернувшись до прикладу, знання адресу, тобто декларативний підхід, передбачає наявність навігатора, що зможе прокласти детальну дорогу по адресі, тобто імперативний підхід.

Занурюючись до програмного коду і залишаючи метафори, можна виділити наступні програмні мови в залежності від їх стилю:

- Імперативні: Java, C++
- Декларативні: HTML, SQL
- Змішанні: C#, JavaScript

Так, можливо помітити, що такі декларативні інструменти як HTML і SQL заявляють про те, що повинно бути зроблено, а не яким чином. Інженер описує бажаний результат, не заглиблюючись в деталі та конкретні кроки інструкції. В той час як мови програмування Java і C++ потребують явного опису інструкції із детальним роз’ясненням усіх кроків, тобто заявляють про те, яким саме чином щось повинно бути зроблено.

2.1.2.2. Парадигми програмування

Парадигми програмування – це шлях за яким класифікуються мови програмування в залежності від їх особливостей і функціоналу. Це спосіб концептуалізації, зо визначає організацію обчислень і структурування роботи, що виконується машиною. Як інженерія програмного забезпечення визначається методологіями, так мови програмування визначаються різними парадигмами. Деякі мови розроблені таким чином, щоб підтримувати одну парадигму (наприклад, Haskell підтримує функціональне програмування), а деякі щоб підтримувати кілька парадигм (наприклад, Java або C++). Тобто, скажімо, програми написані на C++ можуть бути чисто процедурними, об'єктно орієнтованими або можуть вміщати у собі симбіоз цих парадигм і саме інженери вирішують яким чином використовувати ці парадигми в залежності від їх потреб.

Загалом виділяють наступні три найбільш поширені парадигми програмування:

- Функціональне програмування – парадигма програмування, стиль якої полягає у створені структур і елементів комп'ютерної програми, що трактують обчислення як оцінку математичних функцій і уникають зміну станів та непостійні або несталі дані.
- Процедурне програмування – парадигма програмування, що походить від структурного програмування, заснованого на концепції виклику процедури. Процедурі, ще відомі як підпрограми або функції, зазвичай містять описаний набір обчислювальних кроків, що потрібно виконати комп'ютеру.
- Об'єктно орієнтоване програмування – парадигма програмування, що базується на концепті об'єктів, які можуть тримати дані у формі полів, часто названими як атрибути, і код у формі процедур, часто названими як методи.

Функціональне програмування передбачає передання даних із однієї функції в іншу для того щоб отримати результат. В цій парадигмі, функції трактуються як дані, що означає, що її можливо використовувати як параметри, передаючи у інші функції; повертати їх як результат виконання інших функцій; будувати функції використовуючи додаткові функції. Функції в функціональному програмуванні повинні бути чистими, вони повинні уникати спільного стану, а побічні ефекті та дані повинні бути незмінними. Чиста функція – це функція, що буде завжди повертати один і той же результати, передаючі до неї одні і ті же вхідні дані, не покладаючись і не залежачи від локального або глобального стану. Спільній стан – це стан, що ділиться між більш ніж однією функцією або більш ніж однією структурою даних. Тобто, із спільним станом, для того, щоб зрозуміти ефект функції, потрібно знати усі деталі спільної змінної, що додає багато складності та дозволяє меншу здатність модульності.

Процедурне програмування – це програмування, що також відоме як лінійне програмування, що слідує top-down підходу, тобто вся інструкція пишеться зверху донизу і виконується комп’ютером таким же чином. Ця парадигма передбачає написання інструкцій, що повідомляють комп’ютеру що саме потрібно робити крок за кроком покладаючись на процедури і підпрограми.

Об’єктно орієнтоване програмування (ООП) – це програмування, що покладається на інкапсуляцію даних і їх поведінку у об’єкти. ООП додатки використовують колекцію об’єктів, які знають як потрібно виконувати певні дії і як взаємодіяти із іншими елементами додатку. Метод у цій парадигмі можна вважати як процедуру із процедурного програмування, але у ООП вона належить до конкретного класу. Інший важливий аспект ООП це класи, які можна вважати шаблонами для об’єктів.

Говорячи про ООП парадигму, варто згадати чотири важливі принципи такої парадигми:

- Наслідування – це відношення між об’єктами, коли один об’єкт наслідує усі поля (або атрибути) і методи іншого об’єкта. Зазвичай перший об’єкт називають батьківським, а другий дочірній. Дочірній об’єкт успадковується від батьківського і має відповідно в собі всі його поля й методи.
- Поліморфізм – це властивість, що дозволяє використовувати одне й те ж саме ім’я для рішення двох або більше схожим, але технічно різних задач. В більш конкретному прикладі в програмуванні, поліморфізм це перевизначення методу при спадкуванні від іншого класу, взаємозамінність об’єктів одного інтерфейсу, пізнє або динамічне зв’язування і перевантаження методів.
- Інкапсуляція – це приховування деталей реалізації класу і відділення його внутрішнього уявлення від зовнішнього. Ця властивість дозволяє закрити доступ до полів і методів і надавати доступ тільки через певні методи.
- Абстракція – це процес узагальнення. Цей принцип дозволяє працювати із об’єктами, не вдаючись у особливості їх реалізації. Тобто здобуваються необхідні властивості і в той же час опускаються несуттєві деталі.

Так як вся методологія будується навколо класів і об’єктів, то об’єкти можуть взаємодіяти між собою і взаємодіяти різним чином:

- Наслідування – відношення, що є одним із принципів ООП, ще відома як “IS A”, тобто взаємодія, яка дозволяє одному класу перейняти функціонал іншого.
- Реалізація – відношення, при якому клас реалізує функціонал, що описується інтерфейсом.
- Асоціація – відношення, при якому об’єкт одного типу якимось чином зв’язаний із об’єктом іншого типу.

- Агрегація – відношення, ще відоме як “HAS A”, при якому об’єкт зберігає у собі інший об’єкт та використовує його для своїх цілей.
- Композиція – більш строгий варіант агрегації, котрий крім того, що зберігає у собі об’єкт, ще й керує усім циклом його життя. Тобто при знищенні об’єкта, який має відношення композиції з іншим об’єктом, знищується і останній.

Поміж іншим, парадигма об’єктно орієнтованого програмування багата на принципи проектування, де найвідомішим й найбільш поширеним є SOLID, що представляє собою акронім для п’яти принципів проектування, що роблять саме проектування більш зрозумілішим, гнучким там легко підтримуючим на великій дистанції. SOLID вміщає у собі наступні концепції й принципи:

- Single-responsibility principle – принцип єдиною відповідальності, говорить про те, що клас повинен мати тільки одну відповідальність і тільки одну причину для того щоб змінюватись.
- Open-closed principle – принцип відкритості-закритості, говорить про те, що сутності повинні бути відкрити для розширення, але закриті для модифікації.
- Liskov substitution principle – принцип підстановки Лісков, говорить про те, що об’єкти класу у програмі повинні мати можливість бути замінені його дочірніми об’єктами без втрати правильності виконання програми.
- Interface segregation principle – принцип розділення інтерфейсу, про те, що багато специфічних інтерфейсів набагато краще за один загальний інтерфейс.
- Dependency inversion principle – принцип інверсії залежностей, говорить про те, модулі вищого рівня не мають мати залежності від модулів нижчого рівня. Залежності у системі повинні будуватись навколо абстракцій, де абстракції не повинні залежати від деталей, а навпаки, деталі повинні залежати від абстракцій.

Загалом існує багато інших принципів і концепцій, але все ж таки SOLID є одним із основоположним і впливом. Серед інших, є сенс згадати про DRY (англ. Don’t Repeat Yourself), KISS (англ. Keep It Stupid Simple), YAGNI (англ. You Aren’t Gonna Need It). [16]

2.1.2.3. Типізація в мовах програмування

Мови програмування за типізацію поділяються на дві великі частини – типізовані і нетипізовані (безтипізові).

В безтипізових мовах усі сутності рахуються як послідовність біт різної довжини і присутня низькорівневим мовам (наприклад, мова асемблеру).

Типізовані в свою чергу поділяються на декілька категорій, де кожна категорія в свою чергу поділяється на два типи:

- Статична або динамічна типізація. Статична визначається тим, що кінцеві типи встановлюються на етапі компіляції програми, а у динамічної типізації всі типи з’ясовуються під час виконання програми.
- Сильна або слабка типізація (строга або нестрога). Сильна типізація виділяється тим, що не дозволяє змішувати у вираження різні типи і не виконує автоматичне неявне перетворення, а слабка навпаки виконує багато неявних перетворень автоматично навіть якщо можлива втрата точності або перетворення неоднозначне.
- Явна або неявна типізація. Явна типізація потребує явного задання типів нових змінних, функцій або аргументів, де як неявна типізація дозволяє перекладати цю задачу не компілятор або інтерпретатор.

Також важливо підкреслити, що усі ці категорії пересікаються й можливе їх змішування у різних варіантах. Кожна типізація може мати як свої недоліки так і переваги над іншою, але останній вибір залежить від поставленої задачі й від шляху її вирішення.

2.1.3. Прикладний програмний інтерфейс

Прикладний програмний інтерфейс або API (англ. Application Program Interface) – це обчислювальний інтерфейс, що визначає взаємодію між декількома деякими програмними сервісами або додатками. Він визначає тип викликів або запитів, які можна здійснювати, яким чином їх можливо здійснювати, формати переданих й прийнятих даних і яких конвенцій слід дотримуватись. Він також може забезпечити функціонал розширення, щоб користувачі могли розширити вже існуючі механізми різними способами та в різній мірі. API може змінювати свої внутрішні деталі, в той час як на інші компоненти, що використовують його, це ніяк не вплине.

Іноді термін API використовується для позначення підмножини програмних об’єктів (код, підкомпоненти, модулі і т.д.), які фактично реалізують інтерфейс певного компонента або системи.

У побудові програм прикладний програмний інтерфейс спрощує саме програмування, абстрагуючи базову реалізацію та лише виставляючи основний функціонал розробнику. Таким чином, наприклад, графічний інтерфейс може надати користувачеві кнопку, яка виконує усі кроки для отримання і виділення нових листів, API ж для введення та виводу файлів може надати розробнику функціонал, який копією файл з одного місця в інше без повного розуміння розробником операцій із файловою системою, що відбувається під капотом.

Дизайн API має значний вплив на його використання. Принцип приховування даних описує роль інтерфейсів програмування як уможливлення модульного програмування, приховуючи конкретні деталі реалізації модулів, щоб користувачі модулів не потребували розуміння складностей всередині самих модулів. Тобто, прикладний програмний інтерфейс намагається надати користувачу лише ті інструменти, які він би очікував. Дизайн API на сьогоднішній день є важливою частиною будь-якої архітектури програмного забезпечення й організації складної програмної системи.

Документація API завжди присутня, якщо інтерфейс є публічним. Вона описує, які послуги пропонує API та як саме користувач може використовувати ці послуги. Тобто, вона спрямована на охоплення всього, що повинен знати клієнт для своїх практичних цілей.

Документація має вирішальне значення для розробки для розробки та обслуговування додатків. Вона часто представлена через систему документації, наприклад Javadoc, яка має послідовний вигляд і структуру. Однак сам міст, що включений в документації, відрізняється. Але це не є обов'язковим правилом.

З метою ясності, документація може містити опис класів, методів, а також типові сценарії використання із фрагментами коду та обґрунтуванням дизайну. Але головне розуміти, що самі деталі роботи API зазвичай опущені. Також дуже часто документація включає у себе обмеження щодо використання API, наприклад, що деякі параметри не можуть бути відсутніми або функція не є безпечною для потоків, тощо.

Зазвичай API поділяється на три типи відкритості:

- Приватний – призначений лише для внутрішнього використання компанії.
- Партнерський – може використовуватись лише деякими конкретними партнерами компанії, яким було видано доступ й дозвіл на використання.
- Публічний – доступний для широкого використання. Зазвичай компанії відкривають свої інтерфейси для того, щоб розробники могли використовувати їх для написання розширення або впровадження продукту компанії якимось дозволенним чином у свій продукт.

Самих типів прикладних програмних інтерфейсів є багата кількість і використовуються вони для різних цілей, серед них: Web API, Remote API, інтерфейс операційних систем, бібліотеки і фреймворки.

2.2. Серверна частина

Перша і головна частину додатку – це є серверна частина, що бере на себе відповідальність за обробку даних, їх зберігання до бази даних, та за усю бізнес логіку сервісу. Таким чином, дана програмна частина виконує і реалізує перший архітектурний шар розподілу відповідальностей.

2.2.1. Структура проекту та шари абстракцій

Структура проекту поділяється на наступні чотири найбільш поширені архітектурні шари (рис. 2.1):

- Application programming interface layer
- Service layer
- Repository layer
- Data access layer

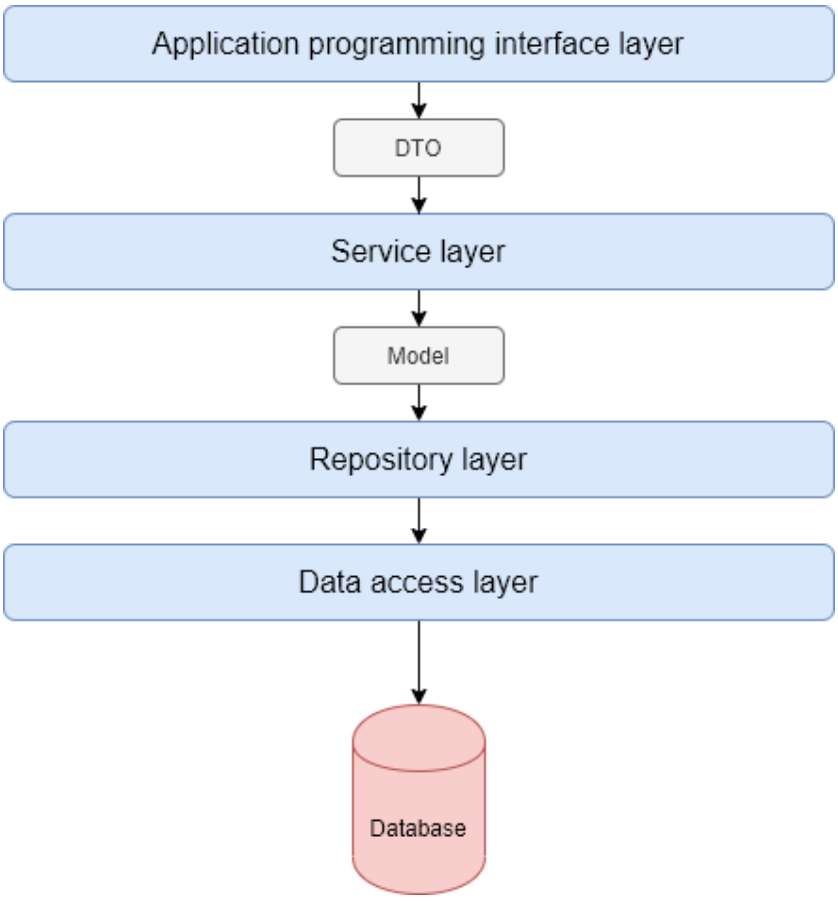


Рис. 2.1 Архітектурні шари

Перший архітектурний шар, що зустрічає запити до програмного додатку, це є Application programming interface шар. Цей шар займається тим, що очікує запити на свої кінцеві точки (англ. endpoint) із зовні додатку та делегує подальшу роботу наступному архітектурному шару. Важливо зауважити, що зазвичай цей шар працює із DTO (англ. Data Access Object) сутностями. DTO займається зберіганням даних і не вміщає у собі ніякою бізнес логіки. Єдина задача DTO – це інкапсулювати дані і переносити або пересилати їх між системами, видаючи тільки ті дані, що того потребують, уникаючи бізнес поведінки і логіки. Мотивацією його використання полягає в тому, що зв'язок між системами зазвичай виконується за допомогою віддалених інтерфейсів, де кожен виклик є дорогою операцією. Таким чином, щоб зменшити кількість викликів інтерфейсу використовують DTO, що агрегує дані в один виклик, які як правилом передаються декількома викликами, що в свою чергу економить час і ресурси системи.

Наступний архітектурний шар, що йде одразу після API – це Service шар. Переваги, що видає цей шар, полягають у тому, що він визначає загальний набір функцій та операцій, що доступні для різних операцій. Тобто, якщо у вас є додаток, який працює із більше ніж одним видом клієнтом, які в свою чергу споживають різну бізнес логіку і мають складні випадки використання, то це саме той випадок використання архітектурного шару Service. Ще ж однією перевагою існування цього рівня полягає у тому, що він інкапсулює бізнес логіку додатку та виклики різних клієнтів. Це допомагає зменшити дублювання коду, оскілки клієнти додатку діляться однаковим загальним функціоналом і також відділяє бізнес логіку системи, що полегшує контроль і підтримку самої бізнес логіки. Також потенційно зменшуються витрати на технічне обслуговування, коли змінюється логіка бізнесу, бо як правило буде потрібно лише змінити сервіс, а не кожного з клієнтів. Що ще варто помітити, так це те, що цей шар зазвичай використовує вже реальні моделі, а не DTO, де моделі являють собою реальні сутності даних, що зберігаються у базі.

Шар, що йде наступним – це Repository, який по суті забезпечує абстрагування даних, задля того щоб система могла працювати з простою абстракцією, маючи лише інтерфейс. Додання, видалення, обновлення або вибірка даних виконується прямо через набір методів або функцій, без потреби конкретної взаємодії із базою даних. Використовуючи цей шар, досягається розділення відповідальності й слабкого зв'язку між шаром бізнесу й шаром роботи із даними та базою, де ці дані зберігаються. Таким чином, система має єдине місце де виконується взаємодія із даними, а слабкий зв'язок дозволяє замінити реальний репозиторій на фальшивий задля реалізації тестів, що дає можливість не використовувати реальну базу даних під час імплементації тестів і їх виконання, а додатковим плюс є легка зміна бази даних, так як система на зав'язується на одній базі, а абстрагується використовуючи інтерфейси. Архітектурний шар Repository доволі схожий на Service шар, але важливо їх розрізняти і розуміти, що Service займається бізнес логікою додатку, в той час як Repository займається доступом до бази даних або будь-якого сховища даних і передачею цих даних від бази до програмного додатку і навпаки.

Наступний шар – це Data Access шар, який, виходячи із своєї назви, виконує роль того, хто напряму займається доступом до бази даних і роботою над даними. Він виступає мостом між бізнес об'єктами й сутностями та самими даними, що зберігаються в базі або будь-якому сховищі. Іноді, Repository шар виконує обов'язок Data Access шару.

Таким чином, кожен архітектурний шар виконує свою задачу, розподіляючи обов'язки й досягаючи слабкого зв'язку, що в свою чергу допомагає й полегшує розробку й подальшу підтримку системи. А можливість змінювати реалізації абстракцій дозволяє безболісно для додатку змінювати деякі частини системи, тобто змінювати або модифікувати систему найшвидшим і найменш ресуро й фінансово затратним шляхом. Такий архітектурний підхід є майже обов'язковим правилом для будь-якої системи,

адже можливість швидко змінюватись та підлаштовуватись під вимоги реалій світу є невід'ємною частино успішності сервісу й проекту в цілому.

2.2.2. Мова програмування Java

Java – це мова програмування загального призначення, заснована на класах та відноситься до об'єктна орієнтованої парадигми. Ця мова із самого початку була призначена для того, щоб розробники мали можливість написати програмний код лише один раз, але могли запускати цей коди де завгодно на будь-якій операційній системі та пристрої. Таким чином компільований код Java може виконуватись і працювати на всіх платформах, що підтримують цю мову без необхідності перекомпіляції.

Java була спочатку розроблена Джеймсом Гослінгом у компанії Sun Microsystems (зараз ця компанія придбана Oracle) і випущена перше в 1995 році. Спочатку Java була призначена для інтерактивного телебачення, але ця ідея була надто просунутою для індустрії цифрового кабельного телебачення на той час. Тоді мова мала назву Oak в честь дерева (з англ. oak - дуб), що стояло біля кабінету Джеймса Гослінга. Пізніше мова прийняла назву Green й через деякий час знайшло свою остаточно назву Java в честь кофейного напою Java, що готується в Індонезії. Гослінг спроектував Java таким чином, щоб синтаксис мови був близьким до синтаксису на той час відомої й поширеної мови C/C++, що в свою чергу було великим плюсом для розробників, адже тоді для них перехід на мову Java було легкою задачею. [2, 3]

Створюючи мову, було п'ять основних цілей і принципів, яким Java повинна була слідувати:

- Вона повинна бути простою, об'єктно орієнтованою і знайомою
- Вона повинна бути міцною і захищеною
- Вона повинна бути архітектурно нейтральною і портативною
- Вона повинна виконуватись із високою продуктивністю
- Вона повинна бути інтерпретованою, потокова й динамічна

2.2.2.1. Версії і видання

Станом на березень 2020 року, Java 8 та 11 підтримуються як версії для довгострокової підтримки. Перелік основних версій Java разом з датами їх випуску:

- JDK 1.0 (Лютий 23, 1996)
- Java SE 6 (Грудень 11, 2006)
- Java SE 7 (Липень 28, 2011)
- Java SE 8 (Березень 18, 2014)
- Java SE 9 (Вересень 21, 2017)
- Java SE 10 (Березень 20, 2018)
- Java SE 11 (Вересень 25, 2018)
- Java SE 12 (Березень 19, 2019)
- Java SE 13 (Вересень 17, 2019)
- Java SE 14 (Березень 17, 2020)

Компанія Sun визначила і підтримує чотири видання Java, що по своїй суті орієнтовані на різні середовища. Sun сегментувала API мови таким чином, щоб вони належали до однієї з платформ. Таким чином існують наступні платформи:

- Java Cards – для смарт карт
- Java Micro Edition (Java ME) – націлена на середовища із обмеженими ресурсами
- Java Standard Edition (Java SE) – націлена на середовища робочих станцій
- Java Enterprise Edition (Java EE) – націлена не багато розповсюджені корпоративні або інтернет середовища

В додаток до цих чотирьох платформ, Sun також видавала таку платформ як Personal Java, що згодом було замінено пізнішою платформою Java Micro Edition. Варто зазначити, що класи в Java API організовані в окремі групи, що називаються пакети (англ. package). Кожен пакет містить набір пов’язаних

інтерфейсів, класів і виключень. Таким чином, кожна платформа має лише свій певний набір пакетів, які призначені для цілей самої конкретної платформи.

2.2.2.2. Синтаксис

Синтаксис Java багато в чому зазнав впливу такої мови як C++. Але у відмінності від C++, що комбінує синтаксис структурованого, загального й об'єктно орієнтованого програмування, Java була створена майже виключно як об'єктно орієнтована мова. Будь-який код пишеться в середині класів, а кожен елемент даних є об'єктом (екземпляром класу), за винятком примітивних типів, які не є об'єктами з міркувань продуктивності. Існують наступі примітивні типи:

- byte: 8-бітний тип, що зберігає цілі числа в діапазоні від -128 до 127 (включно)
- short: 16-бітний тип, що зберігає цілі числа в діапазоні від -32768 до 32767 (включно)
- int: 32-бітний тип, що зберігає цілі числа в діапазоні від -2^{31} до $2^{31} - 1$ (включно)
- long: 64-бітний тип, що зберігає цілі числа в діапазоні від -2^{63} до $2^{63} - 1$ (включно)
- float: 32-бітний тип, що зберігає дійсні числа
- double: 64-бітний тип, що зберігає дійсні чиста
- boolean: булевий тип, що може зберігати лише два значення true (істина) або false (неправда)
- char: тип даних, що зберігає єдиний 16-бітний символ Unicode. Він має мінімальне значення "\u0000" (або 0) і максимальне значення "\uffff" (або 65535 включно).

Доречно додати, що на сьогоднішній день, Java має обгортки у виді класів і об'єктів для кожного із примітивного типу.

У відмінність від C++, Java не підтримує перевантаження оператора або багаторазове успадкування класів, хоча для інтерфейсів підтримується багатократне успадкування.

Java використовує коментарі, що схожі на коментарі C++. Існує три типи коментарів: одностроковий коментар, що позначається двома слешами “//”, багатостроковий коментар, що відкривається із “/*” і закривається за допомоги “*/” і Javadoc тип, що відкривається із “/**” і закривається із “*/”. Javadoc стиль коментарю дає можливість використовувати функціонал Javadoc щоб автоматично створювати документацію для програм які можуть бути підхоплені іншими інструментами розробки.

Вихідні файли повинні бути названі загальнодоступним (англ. public) класом, що вони містять і закінчуватись суфіксом “.java” (наприклад, файл Application.java містить клас Application). Спочатку цей файл компілюється у байт-код, використовуючи компілятор, що створює файл із суфіксом “.class” (наприклад, Application.class) і після чого цей клас можливо бути запустити й виконати. Говорячи про рівні доступу, в Java існує всього чотири рівні доступу, що позначаються наступними ключовими словами:

- public: метод або клас може використовуватись будь-ким і будь-де
- protected: метод або клас може використовуватись лише самим класом, класами в одному пакеті, дочірніми класами в одному або іншому пакеті
- default (не вказується): метод або клас може використовуватись лише самим класом, класами в одному пакеті або дочірніми класами із одного пакету
- private: метод або клас може використовуватись в рамках самого класу

Таким чином, щоб більш детально роздивитись призначення кожного рівня доступу, можна звернутися до таблиці (таблиця 2.1)

Таблиця 2.1

Модифікатори доступу

Модифікатор	Клас	Пакет	Дочірній клас (один пакет)	Дочірній клас (інший пакет)	Будь-де
public	+	+	+	+	+
protected	+	+	+	+	-
default (не вказується)	+	+	+	-	-
private	+	-	-	-	-

Ключове слово “static” позначає статичний метод, що асоціюється тільки із класом, а не конкретним екземпляром класу. Тільки статичні методи можуть викликатись без потрібності у створення об’єкту. Статичні методи не можуть використовувати інші нестатичні методи. Ключове слово “void” вказує на те, що метод не повертає ніякого значення тому, хто викликає метод.

2.2.2.3. Java JVM і байткод

Java Virtual Machine (JVM) – це віртуальна машина, що дає можливість комп’ютеру запускати програмний код, написаний на Java так само як і інший код, щоб був скомпільований у Java байткод. JVM описана специфікацією, яка потрібна для реалізації самої віртуальної машини. Така наявність специфікації забезпечує сумісність написаних на мові Java програм у різних реалізаціях віртуальної машини. Еталона реалізація JVM специфікації розроблена проектом OpenJDK і є доступною у відкритому доступі (англ. open source project), включаючи JIT (Just-In-Time) компілятор під назвою Hot Spot. Комерційно підтримувані версії Java, доступні від корпорації Oracle, базуються на виконанні OpenJDK.

Java байткод – це набір інструкцій, що використовуються для виконання у віртуальній машині Java. Проте розробнику програмного забезпечення не потрібно знати і розуміти Java байткод. Але варто мати деяке розуміння про архітектуру і роботу самого байткоду, адже знання того, як щось працює під капотом на нижчому рівні, підвищує кваліфікованість й майстерність на

вищому рівні. Мета байткоду – це бінарна сумісність. Кожна конкретна операційна система потребує власної реалізації віртуальної машини Java. Проте ці JVM інтерпретують байткод семантично однаково, хоча реальна реалізація може бути різною. Тобто, інструкції діють на наборі загальних абстрагованих типів даних, а не на рідних типах даних набору інструкції будь-якої операційної системи. Важливо зауважити, що додаткові витрати на інтерпретацію вихідного коду в байткод і тільки потім байткоду в машинні інструкції майже завжди уповільнюють процес і систему. Для прискорення такої компіляції, було введено JIT компіляцію з раннього етапу.

Just-in-time компіляція (або компіляція на “льоту”) – це та компіляція, що передбачає компіляцію програмного коду не перед запуском програми, а під час виконання і роботи самої програми. Таким чином, така компіляція допомагає компілювати байткод шляхом компіляції самого байткоду в машинний код безпосередньо під час роботи програми. Такий підхід є поєднанням двох традиційних підходів перекладу у машинний код – Ahead-of-time (AOT) компіляція та інтерпретація, поєднавши як переваги так і недоліки обох. Загальна реалізація компіляції JIT полягає у тому, щоб спершу виконати AOT компіляцію у байткод, а потомім виконати компіляцію JIT у машинний код (динамічна компіляція), замість інтерпретації байду. Це покращує ефективність виконання порівняно із інтерпретацією. JIT компілятори перекладають код постійно, як і інтерпретатори, але кешування скомпільованого коду мінімізує відставання на майбутнє виконання того ж году під час даного запуску.

2.2.2.4. Система автоматичного управління пам’яттю

Мова Java використовує автоматичне збирання сміття (англ. garbage collection) для керування пам’яттю у життєвому циклі об’єкту. Розробник програмного коду визначає сам коли об’єкти повинні бути створені, коли як Java під час роботи програми відповідає за відновлення пам’яті, коли об’єкти більше не використовуються. Як тільки жодного посилання на об’єкт не

залишається, недосяжні об’єкти стають у чергу на звільнення пам’яті збірником сміття. Тим не менш, все ж таки можливий такий сценарій витоку пам’яті, якщо код розробника містить посилання на об’єкт, який більше не потрібен, але так стається, що він зберігається у контейнері.

Однією з ідей, що стоять за автоматичним управлінням пам’яттю, є те, що розробники позбавляються тягаря необхідності виконувати ручне управління пам’яттю. У деяких мова пам’ять для створення об’єктів неявно виділяється із стеку або ж явно виділяється та витісняється з купи. В останньому випадку відповідальність за управління пам’яттю покладається на розробника і при неправильному управлінні можливий витік пам’яті. Якщо програма намагається отримати доступ або перемістити пам’ять, яка вже була виділена й занята, результат є не визначеним і віжко передбачити, і програма, скоріш за всього вийде із ладу. Варто підмітити, що автоматичне збирання сміття не перешкоджає витоку логічної пам’яті, тобто тої, де на пам’ять ще посилаються, але вже ніколи не використовується.

Збір сміття може відбуватися в буд-який час. В ідеалі, звичайно, це відбувається, коли програма не працює. Але він гарантовано спрацьовує, коли в купі недостатньо вільної пам’яті для виділення на новий об’єкт, що свою чергу може спричинити миттєву зупинку програми. Явне управління пам’яттю неможливе в Java.

2.2.2.5. Документація

В Java присутня система автоматичної генерації документації, що називається Javadoc. Ця система має можливість генерувати документацію у HTML формат використовуючи вихідний код Java. Хоча Javadoc дає можливість писати документацію прямо в програмному коді, проте це ніяк не позначається на ефективності виконання програми, адже Javadoc коментарі видаляються під час компіляції. До використання Javadoc генерації, зазвичай розробники писали окрему документацію для програмного забезпечення, але таким чином набагато складніше тримати таку документацію в синхронізації з

самим програмним забезпеченням. Javadoc використовувався Java з першого випуску і зазвичай оновлювався після кожного нового випуску.

Саме використання Javadoc схоже на багатостроковий коментар, але замість відкриваючого “/*” використовується наступний відкриваючий тег “/**” і закривається із “*/”. Перший параграф документації є описом метода, для якого пишеться документації. Після опису, наступним параграфом додається різна кількість описуючих тегів, такі як:

- “@param” – описує параметри метода
- “@return” – описує те, що повертає метод
- “@throws” – описує виключення, які метод може кинути
- інші теги (наприклад “@see”)

2.2.2.6. Бібліотеки класів

Java Class Library – це стандартна бібліотека, що спеціально розроблена для підтримки розробки на мові Java. Вона контролюється Oracle у співпраці із іншими через програму Java Community Process. Компанії і індивідуальні учасники програми мають можливість вплинути на проектування й розробку API. Java Class Library бібліотека містить різний функціонал, до якого входять основні бібліотеки, до яких належать:

- IO/NIO
- Networking
- Reflection
- Concurrency
- Generics
- Scripting/Compiler
- Функціональне програмування (Lambda, Streaming)
- Бібліотеки колекцій, що реалізують такі структури даних як list, dictionary, tree, set, queue, double-ended queue, stack
- Обробка XML (Парсинг, Трансформація, Валідація)

- Security
- Internationalization і localization бібліотеки

Інтеграційні бібліотеки, які дозволяють розробникам програм спілкуватися із зовнішніми системами. До таких бібліотек належать:

- The Java Database Connectivity (JDBC) для доступу до баз даних
- Java Naming and Directory Interface (JNDI) для пошуку і виявлення
- RMI і CORBA для розробки розподілених додатків
- JMX для управління та контролю програм

Бібліотеки для інтерфейсу користувача, до складу яких входять:

- Abstract Window Toolkit (AWT), що забезпечує компоненти для користувацького інтерфейсу
- Swing бібліотеки, що побудовані на AWT
- APIs для захоплення, оброки та відтворення аудіо
- JavaFX для створення настільних програм

Та також інший набір бібліотек, серед яких:

- Плагіни, що дозволяють запускати аплети у веб браузерах
- Java Web Start, що дозволяє ефективно використовувати програми Java для кінцевих користувачів через інтернет
- Реалізація віртуальної машини
- Ліцензування і документація

2.2.3. Інструмент автоматичної збірки Gradle

Збірка – в розробці програмного забезпечення є процесом конвертації вихідного коду, що був написаний розробником, у окремий елемент програмного забезпечення, який можна запустити на комп’ютері й отримати результат виконання програмного додатку.

Автоматична збірка – це процес автоматичного створення програмної збірки і пов’язаними з ними процесами, включаючи: компіляція вихідного коду в машинний код, складання коду, запуск тестів, розгортання програм на

платформах, написання документації. Таким чином поліпшується якість системи, прискорюється процес компіляції і розгортання проекту, позбавляє від не потрібних дій, позбавляє від прив'язки до людини, вводить введення історії збірок і релізів, економить гроші і час.

Gradle – це система для автоматичної збірки проекту з відкритим доступом до коду, що була створена використовуючи концепти Apache Ant і Apache Maven й ввела орієнтування на мову Groovy, замість використання XML описів. Gradle використовує спрямований ациклічний граф, щоб визначити порядок виконання завдань.

Gradle був розроблений для багатопроектних систем, які в свою чергу мають можливість стати досить великими. Він підтримує інкрементальну збірку проекту, інтелектуально визначаючи, які частини дерева збірки актуальні і будь-яке завдання, залежне лише від цих частин, не потрібно повторно виконувати. [10]

2.2.4. Фреймворк Spring

Spring фреймворк – це фреймворк з відкритим доступом до коду, що використовується для створення додатків і як контейнер інверсії контролю для Java платформи. Хоча цей фреймворк не нав'язує жодної конкретної моделі програмування, він став популярний у спільності Java як доповнення або навіть заміна моделі Enterprise JavaBeans (EJB).

Від конфігурації до безпеки, від веб-додатків до великих даних (англ. big data) – незалежно від інфраструктурних потреб програми, Spring фреймворк допоможе побудувати будь-яку систему. До Spring входять такі проекти і окремі фреймворки як: Spring Framework, Spring Boot, Spring Data, Spring Cloud, Spring Security, Spring Session, Spring Integration, Spring HATEOAS, Spring REST, Spring Batch, Spring LDAP, Spring Mobile і багато інших продуктів. Spring робить програмування на Java швидшим, легшим та безпечнішим для всіх. Spring зосереджується на швидкості, простоті та продуктивності, що зробило його найпопулярнішим на платформі Java. [1]

2.2.4.1. Core container модуль

Центром Spring Core є контейнер інверсії контролю (англ. inversion of control), що забезпечує послідовний спосіб налаштування та управління об’єктами Java за допомогою функціоналу рефлексії. Контейнер відповідає за управління життєвими циклами конкретних об’єктів: створення об’єктів, виклик їх методів та налаштування цих об’єктів шляхом їх з’єднання.

Об’єкти, що були створені контейнером, також називаються керованими об’єктами або бінами (англ. bean). Контейнер можна налаштувати, завантаживши XML файли або використавши анотації Java на класах конфігурації. Ці джерела даних містять визначення бінів, які надають інформацію, що необхідна для створення самих бінів.

Об’єкти можливо отримати за допомогою пошуку залежностей або ін’єкції залежностей. Пошук залежностей (англ. dependency lookup) – це шаблон, коли користувач просить контейнер видати йому об’єкт з конкретним ім’ям або об’єкт певного типу. Ін’єкція залежностей (англ. dependency injection) – це схема, коли контейнер передає об’єкти за назвою іншим об’єктами через конструктори, атрибути або спеціальні методи.

У багатьох випадках не потрібно використовувати контейнер під час використання інших частин Spring фреймворку, проте його використання полегшує налаштування програми, забезпечуючи спеціальний механізм для налаштування самих програм та інтеграції майже з усіма середовищами Java, від невеликих додатків до великих корпоративних програм.

2.2.4.2. Data access модуль

Data access модуль вирішує загальні проблеми з якими стикаються розробники під час роботи над додатком із базами даних. Підтримка надається для всіх поширених систем доступу до даних, серед яких: JDBC, Java Persistent API, Hibernate, Apache OJB та інші.

Підтримуючи велику кількість технологій, що надають доступ до бази даних, Spring впроваджує такий функціонал, як управління ресурсами для автоматичного придбання та звільнення ресурсів бази даних; обробка винятків; участь у транзакціях; розгортання ресурсу для отримання об'єктів бази даних із пулу підключень; абстракція для обробки великих бінарних об'єктів (англ. binary large object).

Разом із управлінням транзакціями, структура доступу до даних пропонує гнучку абстракцію для роботи з даними й доступу до них. Spring не пропонує загального інтерфейсу доступу до даних, натомість повна потужність підтримуваних інтерфейсів зберігається недоторканою.

Варто згадати про такий фреймворк як Spring Data – додатковий зручний механізм для взаємодії з сутностями бази даних, організації їх в репозиторії, вилучення даних і модифікація. Загалом для цього буде достатньо лише оголосити інтерфейс і методу в ньому, без імплементації, а Spring Data зробить далі все сам.

Основне поняття в Spring Data – це є репозиторій. Spring забезпечує деяку кількість вбудованих інтерфейсів. Усе що залишається зробити користувачу це лише створити інтерфейс під існуючу сутність у системі і унаслідувати один із вбудованих інтерфейсів Spring Data. Таки чином можливо використовувати наданий функціонал на існуючих в системі сутностях. В мінімальному вигляді це CRUD операції, що дають можливість створити, прочитати, оновити й видалити сутність.

Spring Data надає функціонал створення запиті до бази даних шляхом побудови виходячи із назви методи і для цього він використовує механізм префіксів. Можливо додати умову для властивостей сутностей і об'єднати їх за допомоги ключових слів "And" або "Or". Сам функціонал доволі гнучкий і дозволяє будувати складні й розширені запити.

Якщо ж функціоналу побудови запитів виходячи із назви методів недостатньо, то присутній фунціонал анотації "@Query". Таким чином додається метод до інтерфейсу і над самим методом пишеться анотація. В тілі

анотації використовується або ж нативний SQL або HQL і записується запит так само, як запит пишеться прямо до бази даних. А передані до метода параметри можливо також використовувати у побудові запитів.

Таким чином досягається повне абстрагування від роботи напряму із базою даних, покладаючи на наданий фреймворком функціонал, полегшуючи і прискорюючи розробку додатку фокусуючись на бізнес логіку.

2.2.4.3. Web (Model view controller) модуль

Модуль забезпечує основні функції інтеграції, орієнтовані на веб-сторінки, такі як функція завантаження файлів з декількома файлів та ініціалізація контейнера інверсії контролю за допомогою слухачів сервлетів та веб-орієнтованого контексту програми.

Міститься імплементація шаблону Model-View-Controller й веб-служби REST для веб-додатків. Забезпечується чіткий поділ між кодом моделі домену та веб формами.

2.2.4.4. Aspect-oriented programming модуль

Аспектно-орієнтоване програмування (АОП) – парадигма програмування, заснована на ідеї розділення функціональності для більшого розбиття програми на модулі.

Мотивація створення окремого модулю або фреймворку для аспектно орієнтованого програмування полягала в переконанні, що слід забезпечити базові функції АОП без особливих складностей в розробці, реалізації чи конфігурації.

Структура Spring AOP заснована на проксі шаблоні і налаштовується під час виконання. Це усуває необхідність кроку компіляції. З іншого боку, перехоплення дозволяє лише публічно виконувати методи на існуючих об’єктах у точці з’єднання (англ. join point).

Spring AOP був розроблений таким чином, що він міг працювати із функціоналом наскрізних питань (англ. cross-cutting concerns) всередині Spring фреймворку. Тобто, будь-який об’єкт, створений і налаштований контейнером, може буди збагачений й модифікований за допомогою Spring Aspect-oriented programming фреймворку.

2.2.4.5. Testing модуль

Модуль підтримує модульне тестування (англ. unit testing) та інтеграційне тестування Spring компонентів за допомогою JUnit або TestNG бібліотек. Він забезпечує послідовне завантаження Spring контекстів та їх кешування. Він також надає макети (англ. mock) об’єктів, які ви можете використовувати для коду у повній ізоляції не чіпаючи інший код, що не потребує тестування.

2.2.4.6. Authentication and authorization модуль

Веб додатки вразливі до загроз оскільки вони піддаються відкритому інтернет світу. Доступ до певних веб сторінок, файлів чи інших секретних ресурсів повинен бути обмежений лише авторизованим користувачем. Звичайно, часто застосовуються декілька шарів безпеки, такі як брандмауер, проксі-сервер, безпека віртуальної машини, тощо. Але, щоб контролювати доступ, також має бути певне обмеження безпеки і на рівні програми або додатку.

Таким чином, для автентифікації і авторизації наявний Spring Security фреймворк, що забезпечує функціонал безпеки для додатків створених за допомогою Spring фреймворку.

За допомоги Spring Security можливо реалізувати такі технології безпеки як LDAP, Single sign-on, Basic й Digest автентифікацію, OAuth2 і багато іншого. Доволі гнучкий функціонал фреймворку дозволяє налаштовувати майже будь-який тип перевірки наявності доступу до ресурсів забезпечуючи таким чином максимальний захист від небажаного доступу.

2.2.4.7. Швидка розробка застосунків

Spring Boot фреймворк – це рішення для створення автономних, виробничих класів Spring додатків, які працюють за правилом “просто запусти”. Він заздалегідь налаштований командою Spring на найкращу конфігурацію та використання платформи Spring і сторонніх бібліотек, щоб користувач мав можливість почати з мінімальною суєтою. Таким чином, більшість програм Spring Boot потребують дуже малої конфігурацію, що дозволяє сфокусуватися на бізнес логіку і на розробку додатку, а не на його налаштування. В доповнення, Spring Boot має вбудований сервер Tomcat або Jetty, що дозволяє зібрати проект і тут же його запустити без потреби установки додаткового серверу для веб додатків.

Таким чином досягається легке розуміння та розробка Spring додатків, підвищується продуктивність та зменшується час, що витрачається на розробку і пропонується простіший спосіб розпочати роботу із додатком заливши автоматично конфігурацію на Spring Boot.

2.2.5. JPA реалізація Hibernate ORM

2.2.5.1. Інтерфейс Java Persistence API

Java Persistence Application Programming Interface – це специфікація, що має відношення до зберігання даних. Не всі об’єкти Java потребують зберігання, проте більшість програм зберігають ключові бізнес об’єкти для подальшого їх використання. Специфікація JPA визначає, які об’єкти слід зберігати у програмних додатках та яким чином це потрібно робити.

Сама по собі специфікація JPA не є інструментом або фреймворком, навпаки, вона визначає набір концепцій, які можуть бути реалізовані будь-яким інструментом або фреймворком. Хоча модель об’єктно-реляційного відображення (англ. object-relational mapping) JPA спочатку базувалась на Hibernate, тим не менш, з тих пір вона розвилась. Так само, JPA специфікація

спочатку була призначена для використання з реляційними, тобто SQL, базами даних, але через деякий час з’явились реалізації специфікація, що використовують нереляційні, або NoSQL, бази даних.

Об’єктно-реляційне представлення – це метод програмування для перетворення даних між несумісними системами за допомогою об’єктно-орієнтованих мов програмування. Фактично, таким чином, створюється віртуальна база даних об’єктів, яку можна використовувати у мовах програмування не розмовляючи напряму із базою даних, а лише через призму використовуваної мови. [11]

2.2.5.2. Реалізація Hibernate ORM

Для підключення бази даних до програм Java зазвичай використовувався інструмент Java Database Connectivity (JDBC). Але це породило чимало проблем і вимагало багатьох строчок коду лише для того, щоб підключитися до бази, забираючи багато часу, уповільнюючи розробку. Такі проблеми породили об’єктно-реляційне представлення, що надзвичайно допомогло розробникам з’єднувати об’єктно-орієнтовані програми з базами даних шляхом швидкої та простої реалізації. Щоб скористатися технологією ORM на Java, доступно багато специфічних фреймворків, і серед усіх них Hibernate ORM, що став найпопулярнішим на платформі Java, пропонуючи безліч функцій для роботи із базами даних.

Основна функція Hibernate ORM – це зіставляти класи Java на таблиці бази даних та типи даних Java на типи даних SQL. Він також використовується для написання запитів до бази даних для роботи із самими даними. Була присутня проблема, яка пов’язана з тим, що об’єкт у об’єктно-орієнтованій мові дотримувався об’єктно-орієнтованого принципу програмування, тоді як об’єкт, який зберігався в базі даних, дотримувався нормалізації, тим самим спричинивши невідповідність. Таким чином, Hibernate ORM дотримується принципів зіставлення об’єктів із бази даних із об’єктами у об’єктно-орієнтованій мові програмування.

Hibernate має власну мову запитів, що по своїй суті є ООП версією SQL мови. Основна відмінність SQL від HQL полягає в тому, що HQL не залежить від бази даних, а значить він добре підходить до використання, адже зміна бази даних не вплине на код і не потрібно бути модифікувати програмний додаток. HQL вважається більш потужним, ніж SQL, оскільки у ньому присутні різні додаткові функції, серед яких такі як динамічне профілювання, розбиття на сторінки, тощо.

Hibernate ORM не тільки вирішує проблему невідповідності, з'єднуючи об'єктно-орієнтовані класи з реляційними базами даним, але також забезпечує додатковий контроль над Java додатком, а також контроль над проектуванням бази даних. Фреймворк є дуже потужним і гнучким інструментом, що дає можливість розробниками створювати від простих до складних програм, витримуючи великі навантаження і полегшуючі комунікацію програмного додатку із базами даних.

2.2.6. База даних MySQL

MySQL – це система управління реляційної бази (англ. relational database management system), що базується на мові Structured Query Language (SQL) і має відкритий доступ до програмного коду. MySQL написаний на C та C ++ та працює на всіх поширених платформах, серед яких Linux, UNIX та Windows.

MySQL базується на моделі клієнт-сервер. Ядром MySQL є сервер MySQL, який усі інструкції і команди бази даних. Сервер доступний як окрема програма для використання в мережевому середовищі клієнт-сервер і як бібліотека, яка може бути вбудована в окремі програми.

Спочатку MySQL був розроблений для швидкої обробки великих баз даних. Хоча MySQL зазвичай встановлюється лише на одній машині, він може надсилати базу даних у декілька локацій, дозволяючи таким чином користувачам мати доступ до неї через різні клієнтські інтерфейси, що відправляють команди SQL на сервер, а потом відображають результат роботи.

MySQL підтримує великі бази даних з мільйонами записів і підтримує безліч типів даних, включаючи такі як float, double, char, varchar, binary, text, blob, date, time і багато інших. Для безпеки використовується система привілей доступу та зашифровану систему паролів й підтримує декілька протоколів, включаючи TCP/IP.

Доречно додати, що користувач має можливість змінити вихідний код, щоб відповідати власним очікування, і за цей рівень свободі не потрібно платити, адже доступ до коду MySQL є вільним. [14]

2.2.6.1. Мова структурованих запитів SQL

SQL (Structured Query Language) – це мова структурованих запитів, що використовується для взаємодії із базами даних. Як будь-яка мова програмування, SQL має свій власний синтаксис і розмітку.

Мова використовує запити, які є істотною операцією у SQL. Запит може бути розбитий на деякі елементи, серед яких:

- Clauses – є складовими компонентами у запитах та інструкціях
- Expressions – можуть виробляти або скалярні значення, або таблиці, що складаються із стовпців і рядків даних
- Predicates – задають умови, які можуть бути оцінені за тризначною логікою (істина, неправда, невизначено) і використовується для обмеження висловлювань
- Queries – витягують дані на основі критеріїв, що задаються
- Statements – можуть мати вплив на схеми та дані, або можуть контролювати транзакції, з’єднання, сесії, тощо
- Insignificant whitespace – зазвичай ігнорується у SQL запитах та використовується для спрощення форматування коду задля більшої читабельності

SQL використовує сукупність операторів, інструкцій ті функцій. Загалом оператори поділяються і складаються із запитів, що поділяються на роботу із

структурою бази (Data Definition Language), роботу із рядочками (Data Manipulation Language), роботу з правами (Data Control Language), роботу з транзакціями (Transaction Control Language).

Для роботи із структурою бази використовуються наступні оператори:

- CREATE – створення структури
- ALTER – зміна структури
- DROP – видалення структури

Для роботи із рядочками:

- INSERT – додання даних
- SELECT – вибір даних
- UPDATE – оновлення або модифікація даних
- DELETE – видалення даних

Для роботи із правами і привілеями:

- GRANT – видача привілеїв
- DENY – заборона використання певних структур
- REVOKE – видалення привілеїв, що були раніше видані

Важливо розуміти, що SQL – це стандарт, що реалізований всіма реляційними базами даних, але кожна база має розширення цього стандарту і має особисту мову для роботи із даними, який зазвичай називають діалектом, що звичайно побудований на основі SQL, але надає більше можливостей для повноцінного програмування і надає можливість спрощувати запити. Серед діалектів існують такі як Transact-SQL, що використовується у Microsoft SQL; PL/SQL, що використовується в Oracle базах; PL/pgSQL, що використовується в PostgreSQL. Тим не менш, усі реляційні бази дотримуються стандарту SQL.

2.3. Клієнтська частина

Друга частину додатку – це є клієнтська частина, що бере на себе відповідальність за користувацький інтерфейс, за ту частину, що бачить користувач коли працює із програмним додатком. Іншими словами це

обгортка над серверною частиною або міст між користувачем та серверною частиною. Тобто, дана програмна частина виконує і реалізує другий архітектурний шар розподілу відповідальностей.

2.3.1. Мова програмування JavaScript

JavaScript – це скриптова мова програмування, що дозволяє реалізовувати складні функції на веб сторінках і коли веб-сторінці потрібно робити більше, ніж просто відображати статичну інформацію, то JavaScript може відображати динамічне оновлення вмісту, інтерактивні карти, анімовану графіку, тощо. Мова є вискорівневою, із JIT компіляцією, динамічною, об’єктно-орієнтованою, прототипною. JavaScript частково підтримує і інші парадигми програмування й реалізує стандарт ECMAScript.

На початку 1995 року Брендан Ейх із компанії Netscape взяв на себе відповідальність за розробку на впровадження нової мови для розробників, що не мали вміння роботи із мовою Java, для того щоб додати підтримку Java в продукт компанії Netscape Navigator. Початкова назва мови була "LiveScript", але її швидко перейменували на "JavaScript", через те, що на той час була поширена мова програмування Java.

Сучасний JavaScript – це безпечна мова програмування, що не надає низькорівневий доступ до пам’яті або процесора, оскільки він був створений для браузерів, які цього не потребують. Можливості значною мірою залежать від середовища, в якому ця мова використовується. Наприклад, Node.js підтримує функції запису й читання файлів, виконувати мережеві запити, тощо.

В браузері із JavaScript можна робити все, що стосується маніпуляції веб-сторінками, взаємодії з користувачем та веб-сервером, наприклад:

- Додавати нові сторінки, змінювати вміст та модифікувати стилістику

- Реагувати на дії користувача, відстежуючи натиск миші, рух курсору, натискання клавіш
- Надсилати запит по мережі на віддалені сервери
- Отримувати та встановлювати cookies, показувати повідомлення
- Зберігати дані на стороні клієнта (використовуючи “local storage”)

JavaScript – це легка інтерпретована мова програмування. Веб браузер отримує програмний код в оригінальній текстовій формі і запускає скрипт. З технічною точки зору, більшість сучасних інтерпретаторів JavaScript фактично використовують техніку, що називається JIT компіляція для підвищення продуктивності. Таким чином, вихідний код JavaScript компілюється у більш швидкому, бінарному форматів під час виконання самого скрипту, щоб його можна було запустити якнайшвидше. Однак JavaScript все ж таки вважається інтерпретованою мовою, оскільки компіляція відбувається під час виконання, а не достроково перед запуском.

Є щонайменше три чудові речі щодо JavaScript, через що мова стала поширеною у розробників програмних додатків: повна інтеграція із HTML і CSS, прості речі робляться просто, підтримка всіх основних браузерів та існує за замовчуванням. JavaScript це єдина технологія, що комбінує перечисленні особливості – і це те, що робить цю мову унікальною.

2.3.2. Мова програмування TypeScript

TypeScript – строго типізована, об’єктно-орієнтована, компільована мова, що був розроблений Андерсом Гейлсбергом у компанії Microsoft. Це є і мова, і набір інструментів, що повністю компілюється у JavaScript. Таким чином, TypeScript це JavaScript із додатковими особливостями, що дозволяє знати лише останню мову з метою використання TypeScript.

TypeScript підтримує бібліотеки, інструменти та фреймворки JavaScript, а скомпільований TypeScript можливо використовувати з будь-якого коду JavaScript. Він працює у будь-якому середовищі, в якому працює JavaScript, на

відмінну від своїх аналогів, для яких потрібна виділена віртуальна машина або спеціалізоване середовище.

TypeScript перевершує його інші аналоги, такі як мови програмування CoffeeScript і Dart, так як TypeScript розширяє JavaScript. В той час як CoffeeScript і Dart – це нові мови самі по собі та потребують спеціального середовища для виконання. Серед переваг, що належать до TypeScript, присутні наступні:

- Компіляція. JavaScript – це інтерпретована мова. Отже, програму потрібно запустити, щоб перевірити, що програмний код є дійсним і валідним. Таким чином можлива велика кількість помилок, які в більшості випадків тяжко знайти не витрачаючи деякий час. В той час як TypeScript допомагає виділити помилки перед запуском скрипту, якщо він знайде якість синтаксичні проблеми.
- Строга статична типізація. JavaScript не є строго типізованою мовою. В той час як TypeScript використовує додатковий функціонал статичної типізації.
- Об’єктно-орієнтоване програмування. TypeScript підтримує такі ООП концепти як класи, інтерфейси на наслідування.
- Підтримка існуючих JavaScript бібліотек

Використовуючи TypeScript, розробник отримує безліч потужних переваг поверх тих переваг, що має JavaScript. Але як і будь-яка мова програмування, TypeScript має свій синтаксис, який потребує додаткових зусиль на початку, проте сам синтаксис майже ідентичний до синтаксису JavaScript мови.

2.3.3. Мова розмітки HTML

HTML (Hypertext Text Markup Language) – це комп’ютерна мова розмітки, розроблена для створення веб сайтів. Потім ці веб сайти мають можливість бути переглянутими будь-ким іншим підключеним до інтернету.

Hypertext – це метод, за допомогою якого користувач має можливість пересуватися по інтернету, натискаючи над спеціальний текст, що має назву гіперпосилання (англ. hyperlink), який переводить на інші сторінки. Гіпер означає нелінійність, тобто користувач може переходити в будь-яке місце, натискаючи на посилання і для цього не існує збірки правил.

Markup – це те, яку роль виконують HTML теги із текстом в середині самих тегів. Теги позначають текст як певний тип тексту (наприклад, параграф).

HTML складається з серії коротких строк коду або блоків, що додані розробником сайту в текстовий формат файлу – тобто з так званих тегів. Потім текст зберігається у форматі HTML і переглядається через веб браузер. Веб браузер, в свою чергу, читає файл і переводить тексту у видиму форму таким чином, яким розробник записав у текстовому файлі. Таким чином, написання HTML передбачає правильно використання тегів для створення сторінок.

HTML теги – це те, що відрізняє звичайний текст, від HTML коду. Вони виглядають як слова, що обгорнути у кутові дужки – “<тег>”. Вони дозволяють використовувати картинки, таблиці та інші особливості під час створення сторінки просто повідомляючи про це веб браузеру. Самі ж теги не відображаються, коли ви переглядаєте їх через сторінку використовуючи браузер, але ефект і функціонал, який вони привносять звичайно присутній.

Таким чином, веб браузери отримують HTML документи із веб сервера або локального сховища та переводять ці документи у мультимедійні веб сторінки. В той час, як користувач буде бачити звичайну сторінку для перегляду, під капотом сторінки буде знаходитись HTML код, що і описує цю звичайну сторінку таким чином, щоб будь-який веб браузер зміг її відобразити.

2.3.4. Мова стилю CSS

CSS (Cascading Style Sheets) – каскадні таблиці стилів, що є мовою дизайну, або мовою стилю, що призначення для спрощення процесу

покращення зовнішнього вигляду веб сторінки. За допомогою CSS, розробник має можливість керувати кольором тексту, стилем шрифтів, інтервалом між абзацами, розмірами та розміщенням стовпів, зображенням, варіаціями відображення для різних пристроїв та розмірів екрану, а також багато інших різноманітних ефектів і функцій.

CSS розроблений таким чином, щоб забезпечити розділення презентаційного шару і контенту. Це розділення покращує розробку, забезпечує більшу гнучкість та забезпечує більший контроль у специфікації характеристик презентаційного шару веб сторінки, вказуючи CSS в окремому файлі, який спеціально створюється для зберігання CSS стилю і для уникання повторюваності при стилізації контенту.

Розділення на шари форматування та контенту також робить можливим подання однієї і тієї ж сторінки в різних стилях для різних методів візуалізації, серед яких друк, голос екран або ж на основі шрифту Брайля (для тактильних пристроїв).

Таки чином, CSS економить час під час розробки, даючи можливість написати CSS один раз і використовувати його на багатьох сторінках. Поліпшує швидкість загрузки сторінки, покращує підтримку коду, дає можливість модифікувати сторінку під різні пристрої – що свою чергу зробило CSS стандартом стилізації контенту.

2.3.4.1. Метамова SASS

SASS (Syntactically Awesome Style Sheets) – це розширення CSS, яке дозволяє використовувати так особливості як змінні, вкладені правила, вбудований імпорт, операторі, наслідування та інший функціонал. Це в свою чергу допомагає організовувати речі і дозволяє швидше створювати таблиці стилів для сторінок. SASS сумісний з усіма версіями CSS.

Веб браузери не розуміють код SASS і не знають, що з ним робити. Таким чином використовується SASS препроцесор, що перекладає код SASS у стандартний CSS. Цей процес дуже схожий на компіляцію, але замість

перекладу з людського читаючого вихідного коду в машиночитаний код, цей процес перекладає з однієї читаної людиною мови на іншу, в даному випадку з SASS на CSS. [17]

2.3.5. Веб-фреймворк Angular

Angular – це веб фреймворк з відкритим кодом на основі TypeScript мові, що розробляється і підтримується компанією Google. Angular був повністю переписаний тою самої командою, яка побудувала AngularJS.

Angular – це платформа і фреймворк для створення односторінкових додатків (англ. single-page application) використовуючи HTML і TypeScript. Сам Angular повністю створений за допомогою TypeScript, реалізуючи основний та додатковий функціонал як набір бібліотеки, які користувач має можливість використовувати у своїх програмах.

Архітектура програми Angular опирається на певні фундаментальні концепти. Основними блоками, для створення додатків виступають модулі, які забезпечують контекст компіляції компонентів. Модулі збирають відповідний код у функціональні набори. Додаток створений на Angular визначається набором цих модулів. У буду-якому програмному додатку є завжди принаймні один кореневий модуль, який дозволяє завантажувати сам додаток і зазвичай присутні ще інші модулі.

Компоненти визначаються представлення даних, що представляють собою набори елементів екрану, які Angular дає можливість вибирати та змінювати відповідно до логіки програми.

Компоненти використовують сервіси, які надають певні функціональні можливості, безпосередньо не пов’язані із представленням даних. Сервіси можуть буди введені в компоненти як залежності, що робить код модульним, багаторазовим та ефективним.

І компоненти, і сервіси – це просто класи, в яких є декоратори, які позначаються їх тип та надають метадані, як повідомляють Angular, як ними користуватися. Метадані класу компонента асоціюють його з шаблоном, який

визначає представлення даних. Шаблон поєднує звичайний HTML із Angular директивами та розміткою прив’язки, що дозволяє фреймворку змінювати HTML, так як потребую користувач, перш ніш відображати. Метадані класу сервісу надають інформацію, яку Angular використовує, щоб зробити ці сервіси доступними для можливості їх ін’єкції у виді залежності.

Компоненти програми зазвичай визначають багато представлень, що розташовуються ієрархічно. Angular надає послугу маршрутизатора, що допомагає визначати шляхи для навігації серед представлень. Маршрутизатор забезпечує складні навігаційні можливості в рамках браузера.

Таким чином, компонент і шаблон визначають повне представлення даних. А використання сервісів, що вводяться за допомогою ін’єкції залежностей, надає можливість функціонально збагачувати додаток, роблячи код модульним та ефективним. [6]

2.4. Комунікація між серверним там клієнтськими частинами додатку

2.4.1. Протокол HTTP

HTTP (HyperText Transfer Protocol) – це набір правил і інструкцій, яких повинен дотримуватися сервер, коли мова йде передачу файлів, зображень, відео, аудіо та інших видів файлів через всесвітню павутину.

Механіка та концепція HTTP включає те, що файли пов’язані з іншими файлами через низку посилань. Таким чином, будь-який пристрів веб сервера містить програму, що призначена для відправки та обробки HTTP запитів. Типовий браузер – це клієнт HTTP, який постійно надсилає запити на серверні пристрої. Користувач вводить URL-адресу, або клацає на посилання, після чого браузер створює HTTP запит і потом надсилає його на сервер, що можна знайти за IP адресою, яка вказана в URL адресі.

Таким чином, за допомогою HTTP протоколу можливо налаштувати комунікацію між клієнтською частиною додатку та серверною. Відбувається

це таким чином, що користувач переглядає клієнтську частину і використовує наданий функціонал, в той час, як сама клієнтська частина в тіні постійно робить HTTP запити на сервер, де сервер обробляє запит й відсилає відповідь, що також обробляється на клієнтський частині. А щоб ця комунікація не була хаотична і мала набір правил і специфікацій – існує API, тобто прикладний програмний інтерфейс, що в випадку із веб додатками ще називають Web API. Використовуючи установлений набір правил серверною частиною, клієнтська частина завжди знає на яку адресу відправляти запити і якої відповіді можливо очікувати. [12]

2.4.2. Протокол WS

WS (WebSocket) – це двонаправлений протокол, який використовує клієнт-сервер зв'язок. Він має можливість зберігати стан, тобто це означає, що з'єднання між клієнтом і сервером будем живим, поки його не припинить будь-яка сторона (клієнт чи сервер). Після закриття з'єднання однією із сторін, з'єднання припиняється з обох кінців.

Візьмемо приклад спілкування клієнт-серверної комунікації, де є веб-браузер як клієнт і є сервер. Кожного разу коли ініціюється з'єднання, здійснюється так зване рукостискання (англ. handshaking) і вирішується створення нового з'єднання, що буде триматися живим до тих пір, поки одна із сторін не припинить цього. Таки чином, після рукостискання HTTP протокол змінюється на WS протокол. Коли з'єднання встановлене і живе, зв'язок відбувається за допомогою того ж каналу з'єднання, поки воно не припиниться однією із сторін.

2.4.2.1. Протокол STOMP

STOMP (Simple Text-Oriented Message Protocol) – це текстово-орієнтований протокол, що визначає сумісний провідний формат, щоб будь-хто із доступних клієнтів STOMP міг спілкуватися із будь-яким брокером

повідомлення STOMP, щоб забезпечити просту та широкоросповсюджену взаємодію повідомлень між різними мовами та платформами.

У взаємодії WebSocket протоколу, STOMP лише згадує кілька конкретних способів обміну повідомленнями між клієнтом та сервером. Він визначає протокол для комунікації клієнта і сервера за допомогою повідомлень. Він не визначає жодних деталей реалізації. Він забезпечує більш високу семантику поверх WS протоколу і визначає кілька типів повідомлень, що використовується у клієнт-серверній комунікації. [18]

2.4.2.2. Бібліотека SockJS

SockJS – це JavaScript бібліотека, яка забезпечує об’єкт для роботи із WebSocket. Ідея полягає в тому, що SockJS намагається спочатку використати нативний функціонал WebSocket для установки комунікації. Якщо це не вдається, він має можливість використовувати різні транспортні протоколи, що стосуються конкретного браузера і представляє їх через абстракції, що схожі на WebSocket.

Таким чином, SockJS надає резервний функціонал у випадку, якщо через якісь причини не вдалося встановити комунікації використовуючи нативний WebSocket.

2.5. Безпека комунікації між серверним та клієнтським частинами додатку

2.5.1. Стандарт JWT

JWT (JSON Web Token) – це відкритий стандарт, який визначає компактний та автономний спосіб безпечної передачі інформації між сторонами у вигляді об’єкту JSON. Цю інформацію можна перевірити і довірити, оскільки вона підписана цифровім ключем. JWT можуть бути підписані за допомогою секретного ключа або пари відкритого і закритого ключів.

Хоча JWT можуть бути зашифровані, щоб забезпечити таємницю між сторонами основний упор іде на підписаних ключем жетонах. Підписані жетони можуть бути перевірені на цілісність контенту, що міститься в ньому, в той час як зашифровані жетони можуть приховати контент від інших сторін. Коли жетони підписуються за допомогою пар відкритих і закритих ключів, підпис також засвідчує, що лише сторона, що тримає приватний ключ, є тією, яка його підписала.

Найпоширеніший сценарій для використання JWT – це авторизація. Після входу користувача кожен його наступний запит буде включати JWT, дозволяючи користувачеві отримувати доступ до ресурсів і служб, що дозволені за допомогою цього жетона. Також частий сценарій – це надійний обмін інформації між сторонами. Оскільки JWT можуть бути підписані (наприклад за допомогою пари відкритих і закритих ключів), то надається впевненість у тому, що той, хто відправляє жетон, є тим, ким він себе заявляє. А можливість перевірити, що вміст ніким не був підроблений лише доповнює наданої безпеки жетоном JWT. [13]

2.5.2. Стандарт OAuth2

OAuth2 – це протокол і система авторизації, яка дозволяє програмам отримувати обмежений доступ до облікових записів користувачів на сервісі HTTP, таких як Facebook, Google, GitHub, тощо. OAuth2 делегує автентифікацію користувача тому сервісу, який розміщує обліковий запис користувача, та авторизуючи сторонні додатки для доступу до облікового запису користувача.

OAuth2 визначає чотири ролі:

- Власник ресурсу – це користувач, який авторизує додаток для доступу до свого обліково запису
- Клієнт – це додаток, який хоче отримати доступ до акаунта користувача. Він повинен бути авторизований користувачем, а

авторизація повинна бути прикладним програмним інтерфейсом того сервісу, до якого клієнт хоче отримати доступ.

- Сервер ресурсів – сервер, що розміщує захищені облікові записи користувачів і захищені ресурси
- Сервер авторизації – сервер, що перевіряє особу користувача, а потом видає жетон доступу до програми

Існує декілька способів або шляхів отримання авторизаційного жетону для подальшого його використання задля отримання захищених ресурсів. Найчастіше використовується Authorization Code, оскільки він є оптимізований для серверних додатків, де вихідний код не публічно відкритий і секретний код, що надає сервер авторизації певного сервісу, має можливість зберігатися в таємниці (рис. 2.2). [15]

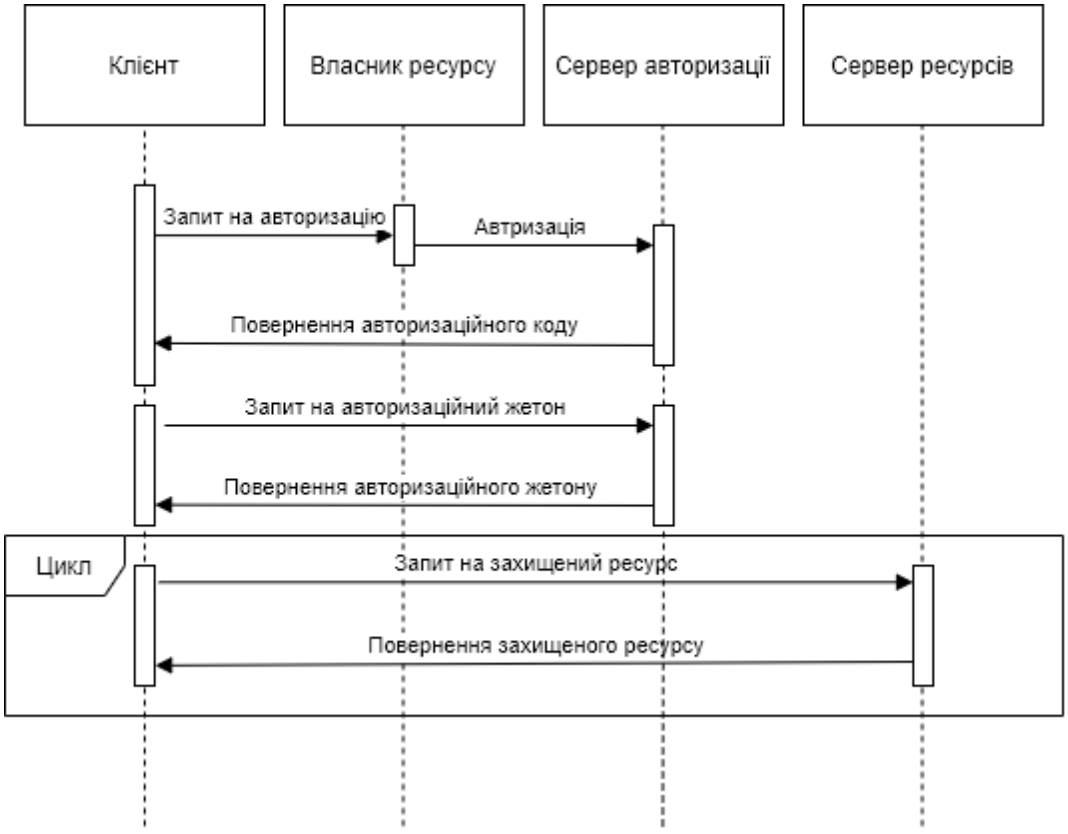


Рис. 2.2 Отримання авторизаційного жетону

ВИСНОВКИ ДО РОЗДІЛУ 2

Таким чином можна побачити, що розподілення програмного додатку на дві частини задля розділення відповідальностей, а саме на клієнтську частину і серверну частину, дає широкий простір для творчості і робить як розробку так і підтримку сервісу доволі гнучкою справою. Сьогоднішній світ технології дає багатий вибір просунутих і професіональних інструментів, що в свою чергу полегшує і прискорює розробку будь-якого додатку.

Огляд сучасних технологій доказав, що сьогодні є можливість створити клієнтський і серверний додатки, що будуть комунікувати між собою, забезпечуючи при тому безпечну комунікацію, адже данні, що користувач буде створювати і використовувати, повинні бути як надійно збереженні, так і мати можливість надійно передаватись від серверної частини до клієнтської та навпаки.

Важливо розуміти, що технології розвиваються кожен день і з ним розвивається швидкість і якість програмної розробки. Тому слід бути в курсі останніх оновлень хоча б тих інструментів, що розробник зазвичай використовує, щоб знати, що змінилося та швидко підлаштуватись під нові особливості.

РОЗДІЛ 3. ІНСТРУКЦІЯ КОРИСТУВАЧА

Першочергова робота користувача із програмним додаток починається із екрану авторизації (рис. 3.1).

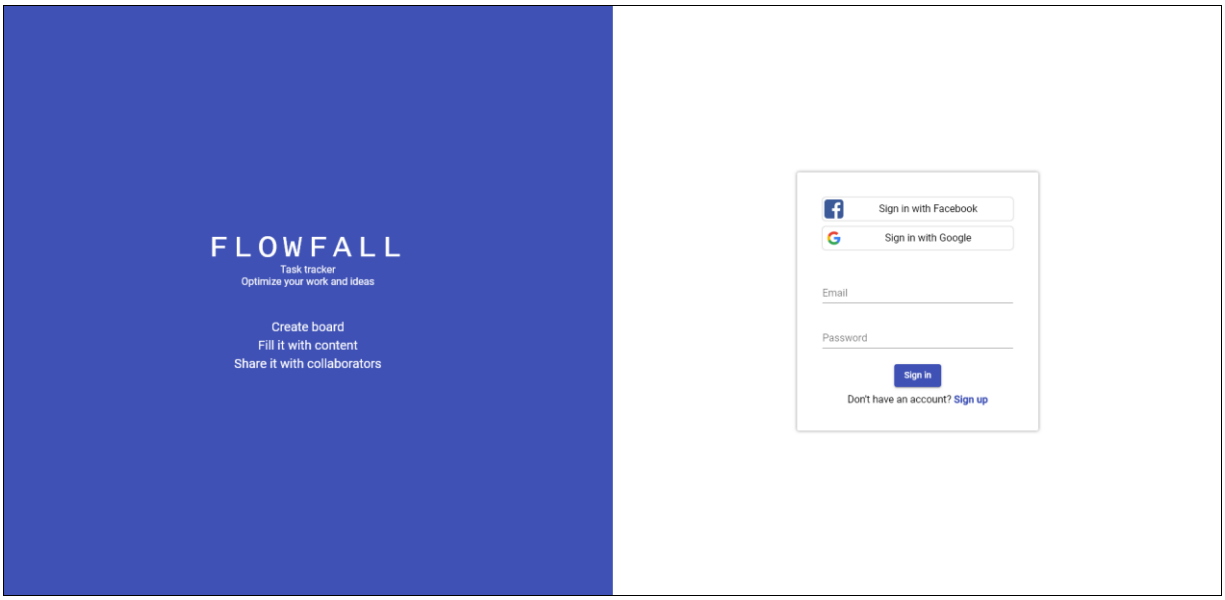


Рис. 3.1 Екран авторизації

На екрані авторизації у користувача є можливість авторизуватися за допомогою логіну і паролю або за допомогою певних соціальних мереж. Також у користувача є можливість створити новий акаунт, змінивши екран авторизації на екран реєстрації натиснувши “Sign up” кнопку (рис. 3.2).

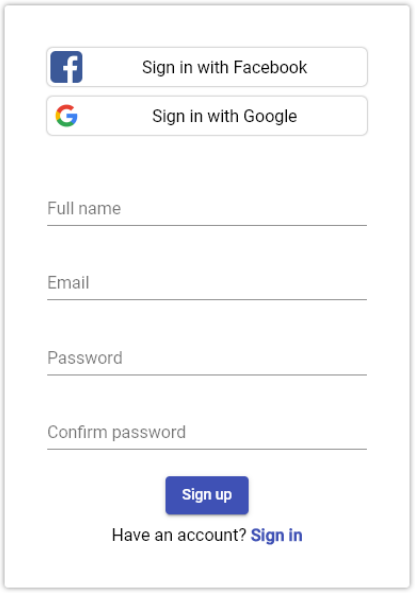


Рис. 3.2 Вікно реєстрації

Вся система авторизації зі сторони користувача виглядає наступним чином, якщо представити її у вигляди діаграми прецедентів (Рис 3.3).

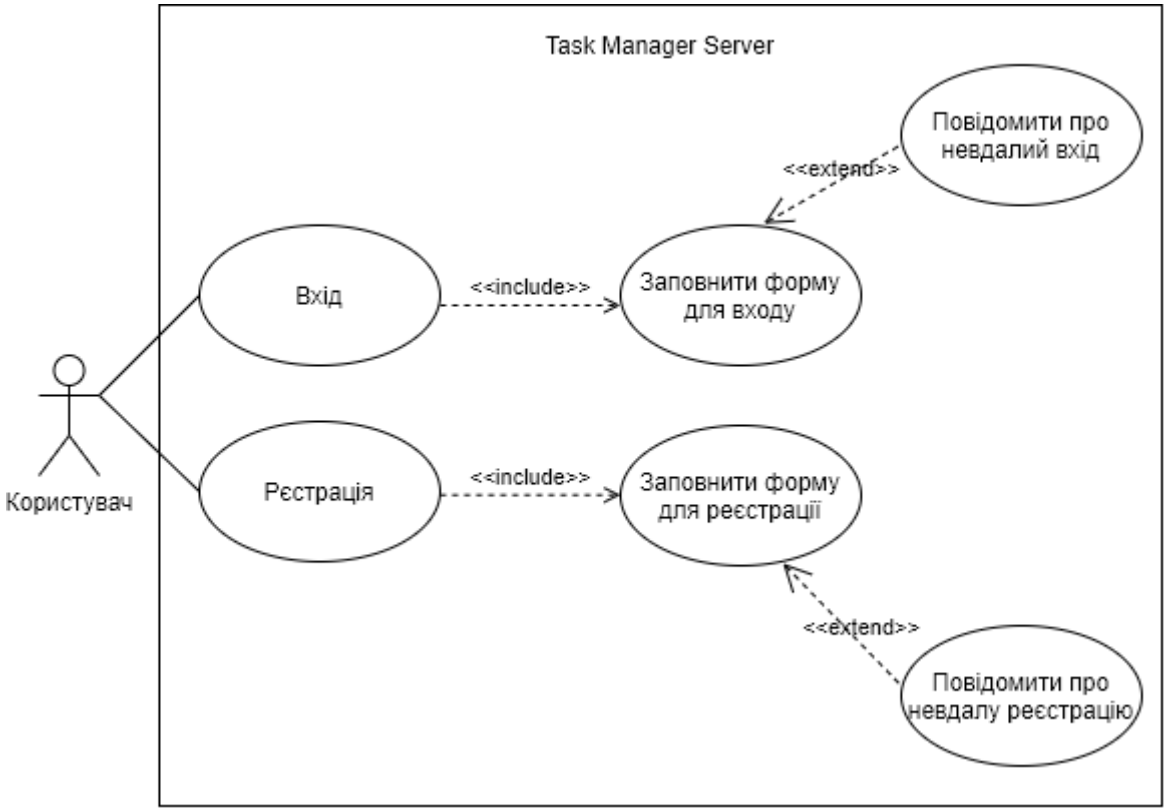


Рис. 3.3 Діаграма авторизації

Після заповнення реєстраційного вікна, на пошту користувача надходить підтвердження у вигляди посилання, переходячи на яке, користувач підтверджує та активує свій акаунт, що дає йому змогу авторизуватися у системі. У випадку, якщо реєстраційна форма була заповнена на правильно (наприклад, обов’язкові поля не були заповнені), то користувач отримає повідомлення про некоректне заповнення. До числа некоректних заповнень входить перевірка на унікальність введеної пошти, тобто, якщо така пошта вже існує, то реєстраційна форма буде недійсною і потребує введення іншої електронної пошти.

Успішно пройшовши авторизацію, користувач потрапляє до простору дошок, де він має змогу створити будь-яку кількість нових дошок для свої цілей або відвідати раніше створену дошку, чи ту, де він співпрацює із іншими користувачами (Рис 3.4).

Рис. 3.4 Простір дошок

Створення нової дошки потребує введення лише назви, після чого користувач потрапляє до головно вікна дошки. Відвідання існуючої дошки також перенаправляє користувача до екрану перегляду самої дошки, де він має можливість почати працювати над своїми задачами (рис. 3.5).

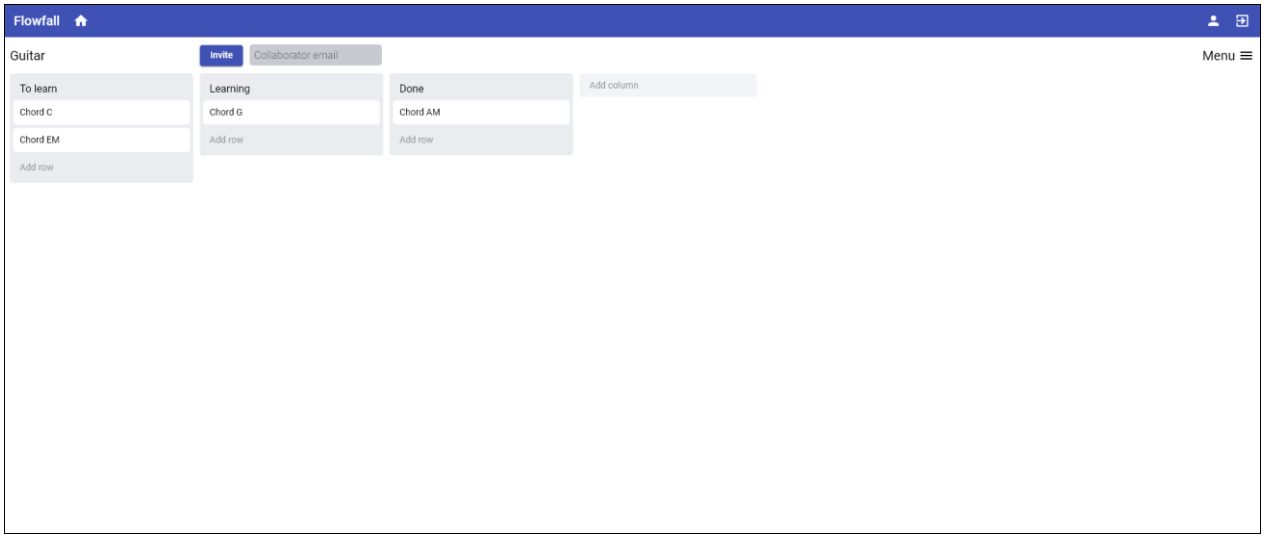


Рис. 3.5 Головне вікно дошки

Потрапивши до головного вікна дошки перед користувачем відкривається основний функціонал додатку – створення і управління задачами. Він має можливість створити будь-яку кількість нових колонок і додати до них будь-яку кількість нових задач із своїми назвами. Створені колонки і задачі можливо перетягувати між собою в будь-якому із наступних варіантів: перетягувати колонки між собою, перетягувати задачі в межах однієї колонки, перетягувати задачі між різними колонками.

До числа функціоналу входить можливість відкрити будь-яку задачу і додати до неї знайдену інформацію або занотувати зауваження (рис. 3.6).

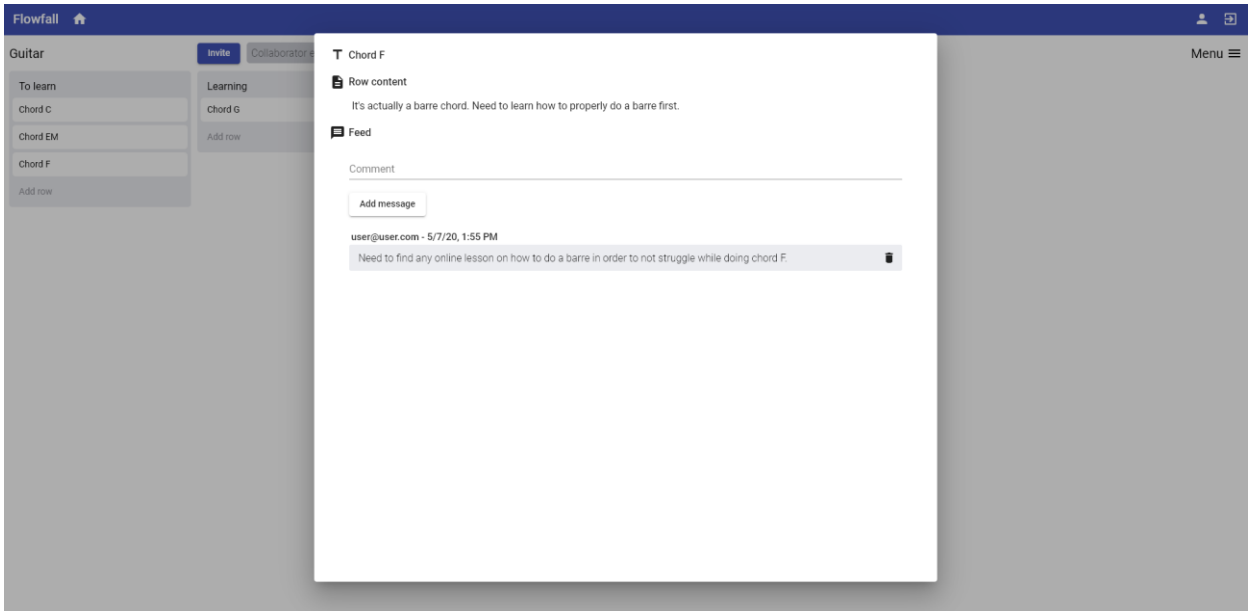


Рис. 3.6 Вікно задачі

Таким чином можливо переглядати додану інформацію, що стосується конкретної задачі, редагувати її або видаляти.

Для співпраці із іншими користувача є можливість додати співучасника до дошки ввівши електрону пошту користувача (рис. 3.7).

Рис. 3.7 Панель додання співучасника

В правому кутку екрана, є можливість відкрити меню дошки, де можна переглянути список співучасників дошки та її власника. А якщо поточний користувач є власником дошки – то він має можливість видаляти співучасників (рис. 3.8).

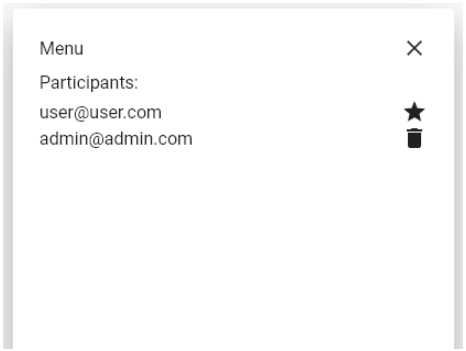


Рис. 3.8 Меню дошки

ВИСНОВКИ ДО РОЗДІЛУ 3

Розроблений програмний додаток, що складається із клієнтської частини і серверної частини, дозволяє користувачу налаштовувати і керувати задачами: створювати дошки; додавати, модифікувати і видаляти як колонки так і самі задачі в колонках; додавати нотатки в задачі; запрошувати і видаляти співучасників для сумісної роботи; та інший функціонал.

Дослідження показало, що таке розділення на клієнтську і серверну частину допомогло поділити відповідальність, спростивши і прискоривши розробку програмного додатку, таким чином наблизившись до створення комфортної функціональності для користувачів.

Користувацький інтерфейс виглядає доволі чисто і мінімалістично з відсутністю зайвого функціоналу, що мотивує користувачів на роботу і фокусує їх увагу на рішенні конкретних задач і питань, не відволікаючи від головної ідеї додатку.

ЗАГАЛЬНІ ВИСНОВКИ

Виконана бакалаврська робота показала, що розподілення відповідальностей допомагає розробляти програмний додаток швидше і більш якісно. Таким чином, клієнтська частина виконується роль обгортки і лиця сервісу, що вітає користувачів і створює міст між користувачем і серверним додаток. Серверний додаток в свою чергу виконує роль головного мозку сервісу, займається обчисленням і збереженням інформації у базу даних. Його фах – це бізнес логіка всієї системи.

Важливо помітити, що кожна частина також розподілена на архітектурні шари абстракції, додаючи ще більший рівень розподілення відповідальностей, полегшуючі розробку і підтримку для розробника програмного забезпечення.

У даній роботі було створено програмний додаток, що займається управлінням задачами шляхом створення дошок. Дошка вміщає у собі деяку кількість колонок, що можуть представляти тип підзадачі або етап її виконання. Кожна колонка вміщає у собі рядки, що представляють функціонал задач як можна додавати, редагувати та видаляти. Для нотування додаткової інформації стосовно задачі є можливість відкрити саму задачу і власне додати потрібну інформацію, що буде збережена, даючи змогу повернутися до неї в будь-який інший час.

Якщо над задачею або завданням повинна працювати група людей задля швидкості і якості його виконання, присутній функціонал співпраці, що дозволяє додавати і видаляти співучасників до дошки і працювати разом над одним питанням.

Таким чином, було створено систему управління задачами, що є комфортною та ефективною як для користувачів, так і для розробників, забезпечуючи комфортну і швидку підтримку, що дає можливість підлаштовуватись під будь-які запити споживачів програмного додатку, що відповідають реаліям сьогодення.

Остаточний варіант додатку у дипломній роботі не є фінальною версію продукту і може бути дороблений, додаючи новий функціонал для поліпшення і модернізації системи. Серед такого функціоналу можливо додання профіля користувача, де можливо зберігати його фотографію із особистими даними; додання можливості відправлення медіа контенту у чат задачі, включаючи веб сайти, картинки і відео матеріал; можливість відновлення тільки-но видаленої колонки або рядка у випадку, якщо видалення було здійснено помилково; додання користувацького посібника для вводу в курс роботи сервісу і його функціональності; зміна дизайну із світлого на темний та навпаки; і багато іншого.

ЛІТЕРАТУРА

1. Craig Walls. Spring in Action, Third Edition / Craig Walls – Wiley India Pvt. Limited, 2011. – 424 p.
2. Herbert Schildt. Java: The Complete Reference, Ninth edition / Herber Schildt – McGraw-Hill Education, 2014. – 1312 p.
3. Joshua Bloch. Effective Java, Second Edition / Joshua Bloch – Addison-Wesley Professional, 2008. – 375 p.
4. Robert C. Martin. Clean Architecture: A Craftsman's Guide to Software Structure and Design / Robert C. Martin – Prentice Hall, 2017. – 432 p.
5. Robert C. Martin. Clean Code: A Handbook of Agile Software Craftsmanship / Robert C. Martin – Prentice Hall, 2009. – 431 p.
6. Angular Docs [Електронний ресурс]. – Режим доступу: https://angular.io/docs/ – Дата доступу: 07.05.2020 – Назва з екрану.
7. Evernote Web Clipper [Електронний ресурс]. – Режим доступу: https://evernote.com/features/webclipper/ – Дата доступу: 07.05.2020 – Назва з екрану.
8. Evernote [Електронний ресурс]. – Режим доступу: https://evernote.com/ – Дата доступу: 07.05.2020 – Назва з екрану.
9. Google Keep [Електронний ресурс]. – Режим доступу: https://keep.google.com/ – Дата доступу: 07.05.2020 – Назва з екрану.
10. Gradle User Manual [Електронний ресурс]. – Режим доступу: https://docs.gradle.org/current/userguide/userguide.html – Дата доступу: 07.05.2020 – Назва з екрану.
11. Hibernate ORM About [Електронний ресурс]. – Режим доступу: https://hibernate.org/orm/ – Дата доступу: 07.05.2020 – Назва з екрану.
12. HTTP Reference [Електронний ресурс]. – Режим доступу: https://developer.mozilla.org/en-US/docs/Web/HTTP – Дата доступу: 07.05.2020 – Назва з екрану.

13. JWT Introduction [Електронний ресурс]. – Режим доступу: https://jwt.io/introduction/ – Дата доступу: 07.05.2020 – Назва з екрану.
14. MySQL Docs [Електронний ресурс]. – Режим доступу: https://dev.mysql.com/doc/ – Дата доступу: 07.05.2020 – Назва з екрану.
15. OAuth Introduction [Електронний ресурс]. – Режим доступу: https://oauth.net/about/introduction/ – Дата доступу: 07.05.2020 – Назва з екрану.
16. Programming paradigm [Електронний ресурс]. – Режим доступу: https://en.wikipedia.org/wiki/Programming_paradigm – Дата доступу: 07.05.2020 – Назва з екрану.
17. Sass Basics [Електронний ресурс]. – Режим доступу: https://sass-lang.com/guide – Дата доступу: 07.05.2020 – Назва з екрану.
18. STOMP Over WebSocket [Електронний ресурс]. – Режим доступу: http://jmesnil.net/stomp-websocket/doc/ – Дата доступу: 07.05.2020 – Назва з екрану.
19. Todoist [Електронний ресурс]. – Режим доступу: https://todoist.com/ – Дата доступу: 07.05.2020 – Назва з екрану.
20. Trello [Електронний ресурс]. – Режиму доступу: https://trello.com/ – Дата доступу: 07.05.2020 – Назва з екрану.

ДОДАТОК 1

Клієнтський та серверний додатки: Task Manager та його безпека

Копії графічних матеріалів

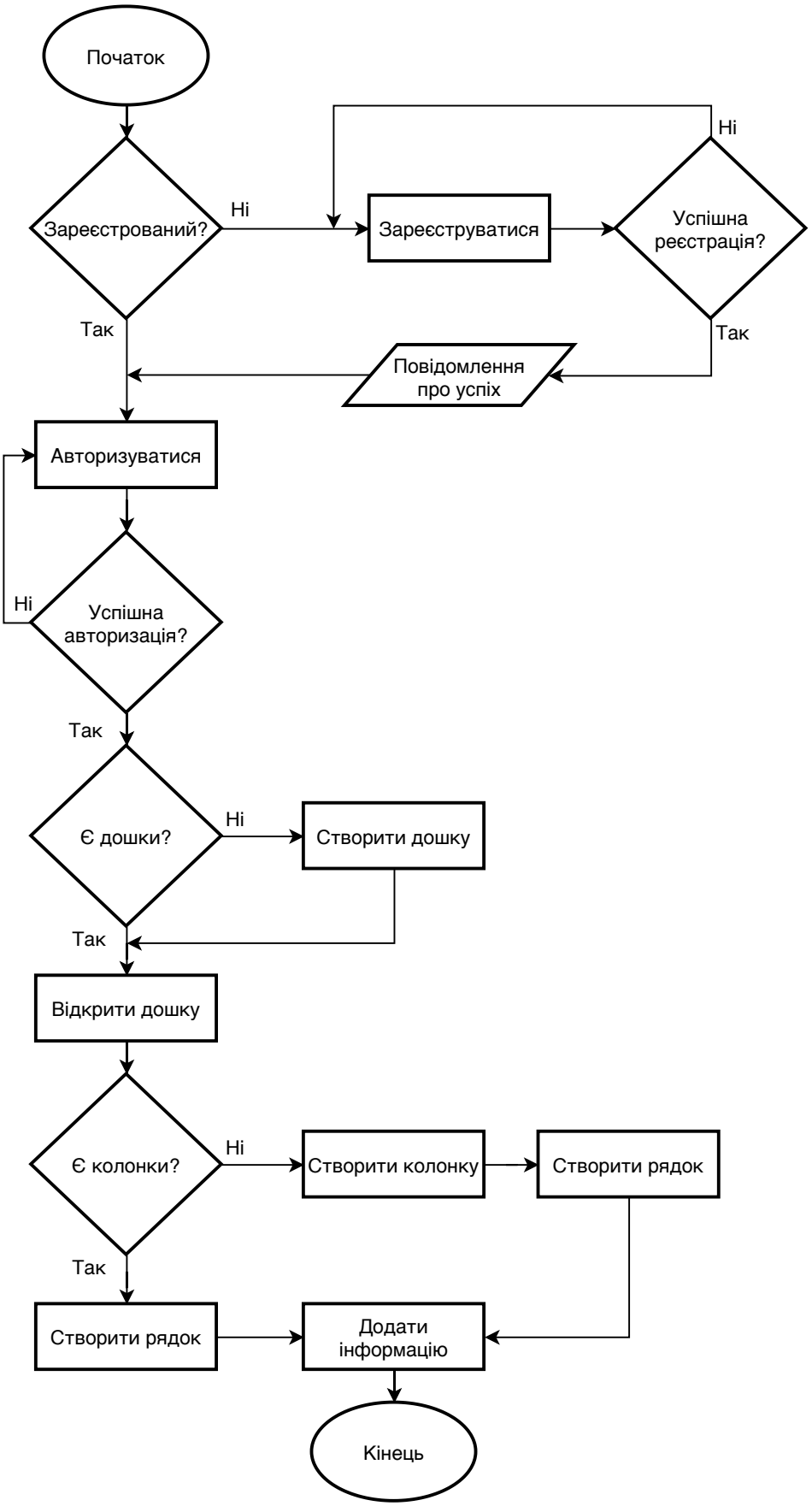
Аркушів 4

Київ 2020 р.

					ІАЛЦ.467100.003 Д1	
					Клієнтський та серверний додатки: Task Manager та його безпека Поведінкова діаграма: діаграма прецедентів	Літ. / Маса / Масш.
Змн.	Арк.	№ докум.	Підпис	Дата		
Розроб.		Рябінін А.Д.				Аркуш 1 / Аркушів 1
Перевір.		Ружевський М.С.				
Т. контр.						«КПІ» ім. Ігоря Сікорського Кафедра ОТ, група ІП-64
Н. контр.		Сімоненко В.П.				
Затверд.						

					ІАЛЦ.467100.004 Д2	
Змн.	Арк.	№ докум.	Підпис	Дата	Клієнтський та серверний додатки: Task Manager та його безпека Структурна діаграма: діаграма пакетів	Літ. / Маса / Масш.
Розроб.		Рябінін А.Д.				
Перевір.		Ружевський М.С.				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоненко В.П.				«КПІ» ім. Ігоря Сікорського Кафедра ОТ, група ІП-64
Затверд.						

					ІАЛЦ.467100.005 Д2	
Змн.	Арк.	№ докум.	Підпис	Дата	Клієнтський та серверний додатки: Task Manager та його безпека Функціональна діаграма: діаграма взаємодії	Літ. / Маса / Масш.
Розроб.		Рябінін А.Д.				
Перевір.		Ружевський М.С.				Аркуш 1 / Аркушів 1
Т. контр.						
Н. контр.		Сімоненко В.П.				«КПІ» ім. Ігоря Сікорського Кафедра ОТ, група ІП-64
Затверд.						



					ІАЛЦ.467100.006 Д4		
Змн.	Арк.	№ докум.	Підпис	Дата	Клієнтський та серверний додатки: Task Manager та його безпека Блок-схема алгоритму	Літ.	Маса / Масш.
Розроб.		Рябінін А.Д.					
Перевір.		Ружевський М.С.				Аркуш 1	Аркушів 1
Т. контр.							
						«КПІ» ім. Ігоря Сікорського Кафедра ОТ, група ІП-64	
Н. контр.		Сімоненко В.П.					
Затверд.							

ДОДАТОК 2

Клієнтський та серверний додатки: Task Manager та його безпека

Лістинг програми

Аркушів 59

Київ 2020 р.

Серверна частина

```
package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.web.bind.annotation.*;
import org.springframework.web.util.UriComponentsBuilder;
import waterfall.flowfall.model.User;
import waterfall.flowfall.model.requests.LoginRequest;
import waterfall.flowfall.model.requests.RegisterRequest;
import waterfall.flowfall.security.AuthFacade;
import waterfall.flowfall.security.AuthProvider;
import waterfall.flowfall.security.AuthUrlBuilder;
import waterfall.flowfall.security.jwt.JwtResponse;

import javax.validation.Valid;

import java.nio.file.AccessDeniedException;

import static org.springframework.http.ResponseEntity.ok;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping(value = "/api/v1/auth")
public class AuthController {

    private AuthFacade authFacade;

    @Autowired
    public AuthController(AuthFacade authFacade) {
        this.authFacade = authFacade;
    }

    @PostMapping(value = "/login")
    public ResponseEntity<JwtResponse> login(@RequestBody LoginRequest loginRequest)
throws AccessDeniedException {
        return ok(authFacade.authenticateAndGetToken(loginRequest));
    }

    @PostMapping(value = "/register")
    public ResponseEntity register(@Valid @RequestBody RegisterRequest
registerRequest) {
        authFacade.register(AuthProvider.LOCAL, registerRequest);
        return new ResponseEntity<>(HttpStatus.OK);
    }

    @GetMapping(value = "/register/verify")
    public ResponseEntity verify(@RequestParam String token, @RequestParam String
redirectUri) {
        return authFacade.verify(token)
                .map(user -> {
                    JwtResponse jwtResponse =
authFacade.authenticateAndGetToken(user);

                    return ResponseEntity.status(HttpStatus.MOVED_PERMANENTLY)
                            .header(HttpHeaders.LOCATION,
AuthUrlBuilder.buildSuccessResponseUrl(jwtResponse, redirectUri))
                            .build();
```

```
                })
                .orElse(ResponseEntity.status(HttpStatus.MOVED_PERMANENTLY)
                        .header(HttpHeaders.LOCATION, redirectUri)
                        .build());
    }
}
```

```
package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.BoardColumn;
import waterfall.flowfall.service.BoardColumnService;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping(value = "/api/v1/boards/{boardId}/columns")
public class BoardColumnController {

    private BoardColumnService boardColumnService;

    @Autowired
    public BoardColumnController(BoardColumnService boardColumnService) {
        this.boardColumnService = boardColumnService;
    }

    @PreAuthorize("@access.require('COLUMN', 'READ', #boardId)")
    @GetMapping(value = "")
    public ResponseEntity<Iterable<BoardColumn>> getColumns(@PathVariable Long
boardId) {
        return new ResponseEntity<>(boardColumnService.findAllByBoardId(boardId),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('COLUMN', 'READ', #boardId)")
    @GetMapping(value = "/{id}")
    public ResponseEntity<BoardColumn> getColumnById(@PathVariable Long boardId,
@PathVariable Long id) {
        return boardColumnService.findById(id)
                .map(column -> new ResponseEntity<>(column, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PreAuthorize("@access.require('COLUMN', 'CREATE', #boardId)")
    @PostMapping(value = "")
    public ResponseEntity<BoardColumn> addColumn(@PathVariable Long boardId,
@RequestBody BoardColumn column) {
        return new ResponseEntity<>(boardColumnService.save(column), HttpStatus.OK);
    }

    @PreAuthorize("@access.require('COLUMN', 'UPDATE', #boardId)")
    @PutMapping(value = "")
    public ResponseEntity<BoardColumn> updateColumn(@PathVariable Long boardId,
@RequestBody BoardColumn column) {
        return new ResponseEntity<>(boardColumnService.update(column),
HttpStatus.OK);
    }
```

```
    @PreAuthorize("@access.require('COLUMN', 'DELETE', #boardId)")
    @DeleteMapping(value = "/{id}")
    public ResponseEntity<Void> deleteColumn(@PathVariable Long boardId,
@PathVariable Long id) {
        return boardColumnService.findById(id)
                .map(boardColumn -> {
                    boardColumnService.delete(boardColumn);
                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
}

package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PostAuthorize;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.Board;
import waterfall.flowfall.model.BoardColumn;
import waterfall.flowfall.model.Row;
import waterfall.flowfall.model.User;
import waterfall.flowfall.service.BoardService;
import waterfall.flowfall.util.SecurityContextUtils;

import java.util.Comparator;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping(value = "/api/v1/boards")
public class BoardController {

    private BoardService boardService;

    @Autowired
    public BoardController(BoardService boardService) {
        this.boardService = boardService;
    }

    @GetMapping(value = "")
    public ResponseEntity<Iterable<Board>> getBoards() {
        User user = SecurityContextUtils.getAuthenticatedUser();
        return new ResponseEntity<>(boardService.findAllByUserId(user.getId()),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('BOARD', 'READ', #id)")
    @GetMapping(value = "/{id}")
    public ResponseEntity<Board> getBoardById(@PathVariable Long id) {
        return boardService.findById(id)
                .map(board -> new ResponseEntity<>(sortByIndex(board),
HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.OK));
    }

    @GetMapping(value = "/collab")
    public ResponseEntity<Iterable<Board>> getCollaborativeBoards() {
        User user = SecurityContextUtils.getAuthenticatedUser();
```

```
        return new
ResponseEntity<>(boardService.findAllCollabBoardsByUserId(user.getId()),
HttpStatus.OK);
    }

    @PostMapping(value = "")
    public ResponseEntity<Board> addBoard(@RequestBody Board board) {
        return new ResponseEntity<>(boardService.save(board), HttpStatus.OK);
    }

    @PreAuthorize("@access.require('BOARD', 'UPDATE', #board.id)")
    @PutMapping(value = "")
    public ResponseEntity<Board> updateBoard(@RequestBody Board board) {
        return new ResponseEntity<>(boardService.update(board), HttpStatus.OK);
    }

    @PreAuthorize("@access.require('BOARD', 'DELETE', #id)")
    @DeleteMapping(value = "/{id}")
    public ResponseEntity<Void> deleteBoard(@PathVariable Long id) {
        return boardService.findById(id)
                .map(board -> {
                    boardService.delete(board);
                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    private Board sortByIndex(Board board) {
        board.getBoardColumns().sort(Comparator.comparingInt(BoardColumn::getIndex));
        board.getBoardColumns()
                .forEach(boardColumn ->
boardColumn.getRows().sort(Comparator.comparingLong(Row::getIndex)));

        return board;
    }
}
```

```
package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.Board;
import waterfall.flowfall.model.User;
import waterfall.flowfall.model.enums.RoleType;
import waterfall.flowfall.service.BoardService;
import waterfall.flowfall.service.UserRoleService;
import waterfall.flowfall.service.UserService;

import java.util.stream.Collectors;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping(value = "/api/v1/boards/{boardId}/collab")
public class CollaboratorController {

    private BoardService boardService;
    private UserService userService;
    private UserRoleService userRoleService;
```

```
    @Autowired
    public CollaboratorController(BoardService boardService, UserService userService,
UserRoleService userRoleService) {
        this.boardService = boardService;
        this.userService = userService;
        this.userRoleService = userRoleService;
    }

    @PreAuthorize("@access.require('BOARD', 'READ', #boardId)")
    @GetMapping(value = "")
    public ResponseEntity<Iterable<User>> getCollaboratorsByBoardId(@PathVariable
Long boardId) {
        return new ResponseEntity<>(userService.findCollaboratorsByBoardId(boardId),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('BOARD', 'READ', #boardId)")
    @GetMapping(value = "/owner")
    public ResponseEntity<User> getOwnerByBoardId(@PathVariable Long boardId) {
        return this.boardService.findById(boardId)
                .map(board -> new ResponseEntity<>(board.getUser(), HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PreAuthorize("@access.require('BOARD', 'INVITE', #boardId)")
    @PostMapping(value = "")
    public ResponseEntity<Void> inviteCollaborator(@PathVariable Long boardId,
@RequestParam String collabEmail) {
        return boardService.findById(boardId)
                .map(board -> {
                    User user = userService.findByEmail(collabEmail).orElse(null);
                    if (user == null) {
                        return new ResponseEntity<Void>(HttpStatus.NOT_FOUND);
                    }

                    if(board.getCollaborators().contains(user)) {
                        return new ResponseEntity<Void>(HttpStatus.OK);
                    }

                    board.addCollaborator(user);
                    Board updatedBoard = boardService.update(board);
                    userRoleService.addRole(updatedBoard.getId(), user.getId(),
RoleType.BOARD_COLLABORATOR);

                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<Void>(HttpStatus.NOT_FOUND));
    }

    @PreAuthorize("@access.require('BOARD', 'INVITE', #boardId)")
    @DeleteMapping(value = "/{collabId}")
    public ResponseEntity<Void> deleteCollaborator(@PathVariable Long collabId,
@PathVariable Long boardId) {
        return boardService.findById(boardId)
                .map(board -> {
                    board.setCollaborators(board.getCollaborators().stream()
                            .filter(collab -> !collab.getId().equals(collabId))
                            .collect(Collectors.toSet())
                    );
```

```
                    Board updatedBoard = boardService.update(board);
                    userRoleService.deleteRole(updatedBoard.getId(), collabId);

                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
}
```

```
package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.*;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.security.AuthFacade;
import waterfall.flowfall.security.jwt.JwtResponse;
import waterfall.flowfall.security.oauth2.OAuth2Facade;
import waterfall.flowfall.security.AuthUrlBuilder;
import waterfall.flowfall.util.CookieUtils;

import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping(value = "/api/v1/oauth2")
public class OAuth2Controller {

    @Value("${security.oauth2.redirectUri}")
    private String oauth2RedirectUri;

    private OAuth2Facade oauth2Facade;
    private AuthFacade authFacade;

    @Autowired
    public OAuth2Controller(OAuth2Facade oauth2Facade, AuthFacade authFacade) {
        this.oauth2Facade = oauth2Facade;
        this.authFacade = authFacade;
    }

    @GetMapping(value = "/code")
    public ResponseEntity oauth(HttpServletResponse response, HttpServletRequest
request,
                                @RequestParam(name="provider") String provider,
                                @RequestParam(name="redirect_uri") String
redirectUri) {
        CookieUtils.addCookie(response, "redirect_uri", redirectUri, 180);
        CookieUtils.addCookie(response, "provider", provider, 180);

        authFacade.verifyProvider(provider);

        return ResponseEntity.status(HttpStatus.MOVED_PERMANENTLY)
                .header(HttpHeaders.LOCATION, AuthUrlBuilder.buildCodeUrl(provider,
oauth2RedirectUri))
                .build();
    }

    @GetMapping(value="/token")
    public ResponseEntity oauth2(HttpServletRequest request, HttpServletResponse
response,
```

```
                                    @RequestParam(name="code") String code) {
        String redirectUri = CookieUtils.getCookie(request,
"redirect_uri").get().getValue();
        String provider = CookieUtils.getCookie(request,
"provider").get().getValue();

        CookieUtils.deleteCookie(request, response, "redirect_uri");
        CookieUtils.deleteCookie(request, response, "provider");

        JwtResponse jwtResponse = oauth2Facade.authenticate(provider, code);

        return ResponseEntity.status(HttpStatus.MOVED_PERMANENTLY)
                .header(HttpHeaders.LOCATION,
AuthUrlBuilder.buildSuccessResponseUrl(jwtResponse, redirectUri))
                .build();
    }
}
```

```
package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.Row;
import waterfall.flowfall.service.RowService;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping("/api/v1/boards/{boardId}/columns/{colId}/rows")
public class RowController {

    private RowService rowService;

    @Autowired
    public RowController(RowService rowService) {
        this.rowService = rowService;
    }

    @PreAuthorize("@access.require('ROW', 'READ', #boardId)")
    @GetMapping(value = "")
    public ResponseEntity<Iterable<Row>> getRows(@PathVariable Long boardId,
@PathVariable Long colId) {
        return new ResponseEntity<>(rowService.findAllByBoardColumnId(colId),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('ROW', 'READ', #boardId)")
    @GetMapping(value = "/{id}")
    public ResponseEntity<Row> getRowById(@PathVariable Long boardId, @PathVariable
Long id) {
        return rowService.findById(id)
                .map(row -> new ResponseEntity<>(row, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PreAuthorize("@access.require('ROW', 'CREATE', #boardId)")
    @PostMapping(value = "")
    public ResponseEntity<Row> addRow(@PathVariable Long boardId, @RequestBody Row
row) {
```

```
        return new ResponseEntity<>(rowService.save(row), HttpStatus.OK);
    }

    @PreAuthorize("@access.require('ROW', 'UPDATE', #boardId)")
    @PutMapping(value = "")
    public ResponseEntity<Row> updateRow(@PathVariable Long boardId, @RequestBody Row
row) {
        return new ResponseEntity<>(rowService.update(row), HttpStatus.OK);
    }

    @PreAuthorize("@access.require('ROW', 'DELETE', #boardId)")
    @DeleteMapping(value = "/{id}")
    public ResponseEntity<Void> deleteRow(@PathVariable Long boardId, @PathVariable
Long id) {
        return rowService.findById(id)
                .map(row -> {
                    rowService.delete(row);
                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }
}

package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.RowMessage;
import waterfall.flowfall.service.RowMessageService;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@RequestMapping("/api/v1/boards/{boardId}/columns/{colId}/rows/{rowId}/messages")
public class RowMessageController {

    private RowMessageService rowMessageService;

    @Autowired
    public RowMessageController(RowMessageService rowMessageService) {
        this.rowMessageService = rowMessageService;
    }

    @PreAuthorize("@access.require('MESSAGE', 'READ', #boardId)")
    @GetMapping(value = "")
    public ResponseEntity<Iterable<RowMessage>> getRowMessagesByRowId(@PathVariable
Long boardId, @PathVariable Long rowId) {
        return new
ResponseEntity<>(rowMessageService.findAllByRowIdOrderByCreatedDesc(rowId),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('MESSAGE', 'READ', #boardId)")
    @GetMapping(value = "/{id}")
    public ResponseEntity<RowMessage> getRowMessageById(@PathVariable Long boardId,
@PathVariable Long id) {
        return rowMessageService.findById(id)
                .map(rowMessage -> new ResponseEntity<>(rowMessage, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
```

```
    }

    @PreAuthorize("@access.require('MESSAGE', 'CREATE', #boardId)")
    @PostMapping(value = "")
    public ResponseEntity<RowMessage> addRowMessage(@PathVariable Long boardId,
@RequestBody RowMessage rowMessage) {
        return new ResponseEntity<>(rowMessageService.save(rowMessage),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('MESSAGE', 'UPDATE', #boardId)")
    @PutMapping(value = "")
    public ResponseEntity<RowMessage> updateRowMessage(@PathVariable Long boardId,
@RequestBody RowMessage message) {
        return new ResponseEntity<>(this.rowMessageService.update(message),
HttpStatus.OK);
    }

    @PreAuthorize("@access.require('MESSAGE', 'DELETE', #boardId)")
    @DeleteMapping(value = "/{id}")
    public ResponseEntity<Void> deleteRowMessage(@PathVariable Long boardId,
@PathVariable Long id) {
        return rowMessageService.findById(id)
                .map(rowMessage -> {
                    rowMessageService.delete(rowMessage);
                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<Void>(HttpStatus.NOT_FOUND));
    }
}

package waterfall.flowfall.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.prepost.PreAuthorize;
import org.springframework.web.bind.annotation.*;
import waterfall.flowfall.model.User;
import waterfall.flowfall.service.UserService;

@RestController
@CrossOrigin(value="*", maxAge = 3600)
@PreAuthorize("@access.isAdmin()")
@RequestMapping(value = "/api/v1/users")
public class UserController {

    private UserService userService;

    @Autowired
    public UserController(UserService userService) {
        this.userService = userService;
    }

    @GetMapping(value = "")
    public ResponseEntity getUsers() {
        return new ResponseEntity<>(userService.findAll(), HttpStatus.OK);
    }

    @GetMapping(value = "/{id}")
    public ResponseEntity<User> getUserById(@PathVariable Long id) {
```

```
        return userService.findById(id)
                .map(user -> new ResponseEntity<>(user, HttpStatus.OK))
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

    @PostMapping(value = "")
    public ResponseEntity<User> addUser(@RequestBody User user) {
        return new ResponseEntity<>(userService.save(user), HttpStatus.OK);
    }

    @PutMapping(value = "")
    public ResponseEntity<User> updateUser(@RequestBody User user) {
        return new ResponseEntity<>(userService.update(user), HttpStatus.OK);
    }

    @DeleteMapping(value = "/{id}")
    public ResponseEntity<Void> removeUser(@PathVariable Long id) {
        return userService.findById(id)
                .map(user -> {
                    userService.delete(user);
                    return new ResponseEntity<Void>(HttpStatus.OK);
                })
                .orElse(new ResponseEntity<>(HttpStatus.NOT_FOUND));
    }

}

package waterfall.flowfall.error;

import org.springframework.http.HttpHeaders;
import org.springframework.http.HttpStatus;
import org.springframework.http.ResponseEntity;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.validation.FieldError;
import org.springframework.web.bind.MethodArgumentNotValidException;
import org.springframework.web.bind.annotation.ControllerAdvice;
import org.springframework.web.bind.annotation.ExceptionHandler;
import org.springframework.web.context.request.WebRequest;
import
org.springframework.web.servlet.mvc.method.annotation.ResponseEntityExceptionHandler;
import waterfall.flowfall.model.dto.ErrorDto;
import waterfall.flowfall.model.dto.ErrorResponseDto;

import java.util.ArrayList;
import java.util.List;

import static org.springframework.http.HttpStatus.*;

@ControllerAdvice
public class ValidationExceptionHandler extends ResponseEntityExceptionHandler {

    @Override
    protected ResponseEntity<Object> handleMethodArgumentNotValid(
            MethodArgumentNotValidException ex, HttpHeaders headers, HttpStatus
status, WebRequest request) {

        List<ErrorDto> errors = new ArrayList<>();

        for(FieldError fieldError: ex.getBindingResult().getFieldErrors()) {
            errors.add(new ErrorDto(fieldError.getField(),
fieldError.getDefaultMessage()));
```

```
        }

        ErrorResponseDto errorResponse =
                new ErrorResponseDto(ex.getLocalizedMessage(),
BAD_REQUEST.getReasonPhrase(), BAD_REQUEST.value(), errors);

        return new ResponseEntity<>(errorResponse, BAD_REQUEST);
    }

    @ExceptionHandler(value = AccessDeniedException.class)
    public ResponseEntity<Object> handleAccessDeniedException(AccessDeniedException
ex, WebRequest request) {
        ErrorResponseDto errorResponse
                = new ErrorResponseDto(ex.getMessage(), FORBIDDEN.getReasonPhrase(),
FORBIDDEN.value());

        return new ResponseEntity<>(errorResponse, FORBIDDEN);
    }

    @ExceptionHandler(value = Exception.class)
    protected ResponseEntity<Object> handleException(Exception ex) {
        ErrorResponseDto errorResponse
                = new ErrorResponseDto(ex.getMessage(),
INTERNAL_SERVER_ERROR.getReasonPhrase(), INTERNAL_SERVER_ERROR.value());

        return new ResponseEntity<>(errorResponse, INTERNAL_SERVER_ERROR);
    }
}

package waterfall.flowfall.error.filter;

import com.fasterxml.jackson.core.JsonProcessingException;
import com.fasterxml.jackson.databind.ObjectMapper;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.web.filter.OncePerRequestFilter;
import waterfall.flowfall.model.dto.ErrorResponseDto;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;

import static org.springframework.http.HttpStatus.FORBIDDEN;
import static org.springframework.http.HttpStatus.INTERNAL_SERVER_ERROR;

public class ExceptionHandlerFilter extends OncePerRequestFilter {

    @Override
    public void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {
        try {
            filterChain.doFilter(request, response);
        } catch (AccessDeniedException e) {
            sendError(response, e.getMessage(), FORBIDDEN.getReasonPhrase(),
FORBIDDEN.value());
        } catch (RuntimeException e) {
            sendError(response, e.getMessage(),
INTERNAL_SERVER_ERROR.getReasonPhrase(), INTERNAL_SERVER_ERROR.value());
        }
    }
```

```
    private void sendError(HttpServletResponse response, String message, String type,
int status) throws IOException {
        ErrorResponseDto errorResponse = new ErrorResponseDto(message, type, status);

        response.setStatus(status);
        response.setContentType("application/json");

        response.getWriter().write(convertObjectToJson(errorResponse));
    }

    private String convertObjectToJson(Object object) throws JsonProcessingException
{
        if (object == null) {
            return null;
        }
        ObjectMapper mapper = new ObjectMapper();
        return mapper.writeValueAsString(object);
    }
}
```

```
package waterfall.flowfall.repository;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Repository;
import waterfall.flowfall.model.enums.EntityType;
import waterfall.flowfall.model.enums.PermissionType;

import javax.persistence.EntityManager;
import javax.persistence.Query;
import java.math.BigInteger;

@Repository
public class RolePermissionRepository {

    @Autowired
    private EntityManager entityManager;

    public boolean hasAccess(long userId, long entityId, EntityType entityType,
PermissionType permissionType) {
        Query query = entityManager.createNativeQuery(
                "SELECT count(*) FROM role_permission" +
                        " JOIN user_role on role_permission.role_id =
user_role.role_id" +
                        " JOIN secured_entity on role_permission.secured_entity_id =
secured_entity.id" +
                        " JOIN permission on role_permission.permission_id =
permission.id" +

                        " WHERE user_id = :userId" +
                        " AND user_role.entity_id = :entityId" +
                        " AND secured_entity.name = :entity" +
                        " AND permission.name = :permission"
        );

        query.setParameter("userId", userId);
        query.setParameter("entityId", entityId);
        query.setParameter("entity", entityType.getLiteral());
        query.setParameter("permission", permissionType.getLiteral());

        return !((BigInteger) query.getSingleResult()).equals(BigInteger.ZERO);
```

```
    }

}
```

```
package waterfall.flowfall.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.authentication.AuthenticationManager;
import
org.springframework.security.authentication.UsernamePasswordAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.context.SecurityContextHolder;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.stereotype.Component;
import waterfall.flowfall.enums.Template;
import waterfall.flowfall.model.GlobalRole;
import waterfall.flowfall.model.User;
import waterfall.flowfall.model.UserProfile;
import waterfall.flowfall.model.VerificationToken;
import waterfall.flowfall.model.enums.UserGlobalRole;
import waterfall.flowfall.model.requests.LoginRequest;
import waterfall.flowfall.model.requests.RegisterRequest;
import waterfall.flowfall.security.jwt.JwtProvider;
import waterfall.flowfall.security.jwt.JwtResponse;
import waterfall.flowfall.security.oauth2.exception.OAuth2AuthenticationException;
import waterfall.flowfall.service.UserService;
import waterfall.flowfall.service.VerificationTokenService;
import waterfall.flowfall.util.EmailUtils;

import java.nio.file.AccessDeniedException;
import java.util.*;

import static waterfall.flowfall.enums.VerifyTemplate.*;


@Component
public class AuthFacade {

    private AuthenticationManager authenticationManager;
    private JwtProvider jwtProvider;
    private UserService userService;
    private VerificationTokenService verificationTokenService;
    private EmailUtils emailUtils;

    @Qualifier("customUserDetailsService")
    @Autowired
    private UserDetailsService userDetailsService;

    @Value("${water.verificationTokenExpirationInMinutes}")
    private int verificationTokenExpirationInMinutes;

    @Value("${water.supportEmail}")
    private String supportEmail;

    @Value("${water.apiUrl}")
    private String apiUrl;

    @Autowired
    public AuthFacade(JwtProvider jwtProvider, UserService userService,
```

```
AuthenticationManager authenticationManager,
                        VerificationTokenService verificationTokenService, EmailUtils
emailUtils) {
        this.authenticationManager = authenticationManager;
        this.jwtProvider = jwtProvider;
        this.userService = userService;
        this.verificationTokenService = verificationTokenService;
        this.emailUtils = emailUtils;
    }

    public JwtResponse authenticateAndGetToken(LoginRequest loginRequest) throws
AccessDeniedException {
        return userService.findByEmail(loginRequest.getEmail())
            .map(user -> {
                authenticate(loginRequest);
                String jwt = jwtProvider.generateJwtToken(user);

                return new JwtResponse(jwt, user.getEmail(), user.isVerified());
            })
            .orElseThrow(() -> new AccessDeniedException("No user with email " +
loginRequest.getEmail()));
    }

    public JwtResponse authenticateAndGetToken(User user) {
        authenticate(user);
        String jwt = jwtProvider.generateJwtToken(user);

        return new JwtResponse(jwt, user.getEmail(), user.isVerified());
    }

    public Authentication authenticate(LoginRequest loginRequest) {
        return authenticateWithCredentials(loginRequest.getEmail(),
loginRequest.getPassword());
    }

    public Authentication authenticate(User user) {
        return authenticationWithoutCredentials(user.getEmail());
    }

    public Authentication authenticate(String email) throws AccessDeniedException {
       return userService.findByEmail(email)
                .map(user -> authenticationWithoutCredentials(email))
                .orElseThrow(() -> new AccessDeniedException("No user with email " +
email));
    }

    public void register(AuthProvider provider, RegisterRequest registerRequest) {
        User user = new User();
        user.setProfile(new UserProfile(registerRequest.getFullname()));
        user.setProvider(provider);
        user.setEmail(registerRequest.getEmail());
        user.setPassword(registerRequest.getPassword());
        user.setGlobalRoles(Collections.singletonList(new
GlobalRole(UserGlobalRole.USER)));
        user.setVerified(false);

        sendVerificationToken(userService.save(user),
registerRequest.getRedirectUri());
    }

    public Optional<User> verify(String token) {
```

```
        return verificationTokenService.findByToken(token)
                .map(foundToken -> {
                    verificationTokenService.delete(foundToken);

                    boolean isExpired = foundToken.getExpirationDate().before(new
Date());
                    if (isExpired) {
                        return Optional.<User>empty();
                    }

                    User user = foundToken.getUser();
                    user.setVerified(true);

                    return Optional.of(userService.update(user));
                })
                .orElse(Optional.empty());
    }

    public boolean verifyProvider(String provider) {
        try {
            AuthProvider.valueOf(provider.toUpperCase());
        } catch (IllegalArgumentException e) {
            throw new OAuth2AuthenticationException("Provider " + provider + " is not
supported currently");
        }

        return true;
    }

    private void sendVerificationToken(User user, String redirectUri) {
        VerificationToken verificationToken =
                new VerificationToken(user, UUID.randomUUID().toString(),
verificationTokenExpirationInMinutes);
        verificationTokenService.save(verificationToken);

        Map<String, String> valuesToBind = new HashMap<>();
        valuesToBind.put(URL.getLiteral(), apiUrl + "/auth/register/verify");
        valuesToBind.put(NAME.getLiteral(), user.getProfile().getFullname());
        valuesToBind.put(TOKEN.getLiteral(), verificationToken.getToken());
        valuesToBind.put(REDIRECT_URI.getLiteral(), redirectUri);

        emailUtils.send(user.getEmail(), supportEmail, "Verification",
                emailUtils.renderTemplate(Template.VERIFY, valuesToBind));
    }

    private Authentication authenticateWithCredentials(String email, String password)
{
        Authentication authentication = authenticationManager.authenticate(
            new UsernamePasswordAuthenticationToken(email, password)
        );
        SecurityContextHolder.getContext().setAuthentication(authentication);

        return authentication;
    }

    private Authentication authenticationWithoutCredentials(String email) {
        Authentication authentication = new
UsernamePasswordAuthenticationToken(email, null, null);
        SecurityContextHolder.getContext().setAuthentication(authentication);

        return authentication;
```

```
    }
}
```

```
package waterfall.flowfall.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.core.userdetails.UserDetails;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.core.userdetails.UsernameNotFoundException;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.User;
import waterfall.flowfall.service.UserService;

@Service
public class CustomUserDetailsService implements UserDetailsService {

    private UserService userService;

    @Autowired
    public CustomUserDetailsService(UserService userService) {
        this.userService = userService;
    }

    @Override
    public UserDetails loadUserByUsername(String email) throws
UsernameNotFoundException {

        User user = userService.findByEmail(email)
                .orElseThrow(() -> new UsernameNotFoundException("User is not found
with email: " + email));

        return UserPrincipal.build(user);
    }
}
```

```
package waterfall.flowfall.security;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.context.annotation.Bean;
import org.springframework.context.annotation.Configuration;
import org.springframework.security.authentication.AuthenticationManager;
import org.springframework.security.authentication.dao.DaoAuthenticationProvider;
import
org.springframework.security.config.annotation.authentication.builders.Authentication
ManagerBuilder;
import
org.springframework.security.config.annotation.method.configuration.EnableGlobalMetho
dSecurity;
import org.springframework.security.config.annotation.web.builders.HttpSecurity;
import
org.springframework.security.config.annotation.web.configuration.EnableWebSecurity;
import
org.springframework.security.config.annotation.web.configuration.WebSecurityConfigure
rAdapter;
import org.springframework.security.config.http.SessionCreationPolicy;
import org.springframework.security.core.userdetails.UserDetailsService;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import
org.springframework.security.web.authentication.UsernamePasswordAuthenticationFilter;
```

```
import waterfall.flowfall.error.filter.ExceptionHandlerFilter;
import waterfall.flowfall.security.jwt.JwtAuthFilter;

import java.util.List;

@Configuration
@EnableWebSecurity
@EnableGlobalMethodSecurity(
    prePostEnabled = true
)
public class SecurityConfiguration extends WebSecurityConfigurerAdapter {

    @Qualifier("customUserDetailsService")
    @Autowired
    private UserDetailsService userDetailsService;

    @Value("${security.allowedApis}")
    private List<String> allowedApis;

    @Override
    protected void configure(AuthenticationManagerBuilder auth) throws Exception {
        auth.authenticationProvider(authProvider());
    }

    @Override
    protected void configure(HttpSecurity http) throws Exception {
        http.cors().and().csrf().disable().authorizeRequests()
                .antMatchers(allowedApis.toArray(new String[0])).permitAll()
                .anyRequest().authenticated()
            .and()

.sessionManagement().sessionCreationPolicy(SessionCreationPolicy.STATELESS);

        http.addFilterBefore(authFilter(),
UsernamePasswordAuthenticationFilter.class);
        http.addFilterBefore(exceptionHandlerFilter(), JwtAuthFilter.class);
    }

    @Bean
    @Override
     public AuthenticationManager authenticationManagerBean() throws Exception {
        return super.authenticationManager();
    }

    @Bean
    public JwtAuthFilter authFilter() {
        return new JwtAuthFilter();
    }

    @Bean
    public ExceptionHandlerFilter exceptionHandlerFilter() {
        return new ExceptionHandlerFilter();
    }

    @Bean
    public DaoAuthenticationProvider authProvider() {
        DaoAuthenticationProvider authProvider = new DaoAuthenticationProvider();

        authProvider.setUserDetailsService(userDetailsService);
        authProvider.setPasswordEncoder(passwordEncoder());
```

```
        return authProvider;
    }

    @Bean
    public BCryptPasswordEncoder passwordEncoder() {
        return new BCryptPasswordEncoder();
    }
}
```

```
package waterfall.flowfall.security.jwt;

import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Qualifier;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.security.authentication.AbstractAuthenticationToken;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.userdetails.UserDetailsService;
import
org.springframework.security.web.authentication.WebAuthenticationDetailsSource;
import org.springframework.web.filter.OncePerRequestFilter;
import waterfall.flowfall.security.AuthFacade;

import javax.servlet.FilterChain;
import javax.servlet.ServletException;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import java.io.IOException;
import java.util.List;

public class JwtAuthFilter extends OncePerRequestFilter {

    private static final Logger logger =
LoggerFactory.getLogger(JwtAuthFilter.class);

    @Autowired
    private JwtProvider jwtProvider;

    @Autowired
    private AuthFacade authFacade;

    @Qualifier("customUserDetailsService")
    @Autowired
    private UserDetailsService userDetailsService;

    @Value("${security.allowedApis}")
    private List<String> allowedApis;

    @Override
    protected void doFilterInternal(HttpServletRequest request, HttpServletResponse
response, FilterChain filterChain) throws ServletException, IOException {

        if (!isAllowedApi(request.getServletPath())) {
            String token =
jwtProvider.getJwtFromHeader(request.getHeader(jwtProvider.JWT_HEADER_NAME));

            if (token == null) {
                throw new AccessDeniedException("Authorization token is not
specified");
```

```
            }

            if (!jwtProvider.validateJwtToken(token)) {
                throw new AccessDeniedException("Authorization token is not valid");
            }

            Authentication auth =
authFacade.authenticate(jwtProvider.getEmailFromJwtToken(token));
            ((AbstractAuthenticationToken) auth).setDetails(new
WebAuthenticationDetailsSource().buildDetails(request));
        }

        filterChain.doFilter(request, response);
    }

    private boolean isAllowedApi(String api) {
        boolean allowed = false;

        for (String allowedApi: allowedApis) {
            if (allowedApi.endsWith("**")) {
                allowed = api.startsWith(allowedApi.substring(0, allowedApi.length()
- 3));
            } else {
                allowed = api.equals(allowedApi);
            }

            if (allowed) {
                break;
            }
        }

        return allowed;
    }
}
```

```
package waterfall.flowfall.security.jwt;

import io.jsonwebtoken.*;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.stereotype.Component;
import waterfall.flowfall.model.User;

import java.util.Date;
import java.util.HashMap;
import java.util.Map;

@Component
public class JwtProvider {

    private static final Logger logger = LoggerFactory.getLogger(JwtProvider.class);

    @Value("${water.jwtHeaderName}")
    public String JWT_HEADER_NAME;

    @Value("${water.jwtSecret}")
    private String jwtSecret;

    @Value("${water.jwtExpiration}")
    private int jwtExpiration;
```

```
    public String generateJwtToken(User user) {
        return Jwts.builder()
                .setClaims(generateClaims(user))
                .setSubject(user.getEmail())
                .setIssuedAt(new Date())
                .setExpiration(new Date((new Date()).getTime() + jwtExpiration *
1000))
                .signWith(SignatureAlgorithm.HS512, jwtSecret)
                .compact();
    }

    public boolean validateJwtToken(String token) {
        try {
            Jwts.parser().setSigningKey(jwtSecret).parseClaimsJws(token);
            return true;
        } catch (SignatureException e) {
            logger.error("Invalid JWT signature -> Message: {} ", e);
        } catch (MalformedJwtException e) {
            logger.error("Invalid JWT token -> Message: {}", e);
        } catch (ExpiredJwtException e) {
            logger.error("Expired JWT token -> Message: {}", e);
        } catch (UnsupportedJwtException e) {
            logger.error("Unsupported JWT token -> Message: {}", e);
        } catch (IllegalArgumentException e) {
            logger.error("JWT claims string is empty -> Message: {}", e);
        }

        return false;
    }

    public String getEmailFromJwtToken(String token) {
        return Jwts.parser()
                .setSigningKey(jwtSecret)
                .parseClaimsJws(token)
                .getBody().getSubject();
    }

    public String getJwtFromHeader(String authHeader) {
        if(authHeader != null && authHeader.startsWith("Bearer ")) {
            return authHeader.replace("Bearer ", "");
        }

        return null;
    }

    private Map<String, Object> generateClaims(User user) {
        Map<String, Object> claims = new HashMap<>();
        claims.put("id", String.valueOf(user.getId()));

        return claims;
    }
}
```

```
package waterfall.flowfall.security.oauth2;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.beans.factory.annotation.Value;
import org.springframework.http.HttpMethod;
import org.springframework.stereotype.Component;
import org.springframework.web.client.RestTemplate;
```

```
import waterfall.flowfall.model.GlobalRole;
import waterfall.flowfall.model.User;
import waterfall.flowfall.model.UserProfile;
import waterfall.flowfall.model.enums.UserGlobalRole;
import waterfall.flowfall.security.AuthFacade;
import waterfall.flowfall.security.AuthProvider;
import waterfall.flowfall.security.AuthUrlBuilder;
import waterfall.flowfall.security.jwt.JwtResponse;
import waterfall.flowfall.security.oauth2.userinfo.OAuth2UserInfo;
import waterfall.flowfall.service.UserService;

import java.util.Arrays;
import java.util.Map;
import java.util.Optional;

@Component
public class OAuth2Facade {

    private RestTemplate restTemplate;
    private UserService userService;
    private AuthFacade authFacade;

    @Value("${security.oauth2.redirectUri}")
    private String oauth2RedirectUri;

    @Autowired
    public OAuth2Facade(UserService userService, AuthFacade authFacade) {
        this.restTemplate =  new RestTemplate();
        this.userService = userService;
        this.authFacade = authFacade;
    }

    public JwtResponse authenticate(String provider, String code) {
        Map map = restTemplate.exchange(AuthUrlBuilder.buildTokenUrl(provider, code,
oauth2RedirectUri),
                HttpMethod.POST, null, Map.class).getBody();
        OAuth2UserInfo userInfo = OAuth2UserInfoFactory.getOAuth2UserInfo(provider,
                restTemplate
                    .exchange(
                        AuthUrlBuilder.buildUserInfoUrl(provider,
map.get("access_token").toString()),
                        HttpMethod.GET, null, Map.class)
                    .getBody()
        );

        Optional<User> optionalUser = userService.findByEmail(userInfo.getEmail());

        if (optionalUser.isPresent()) {
            return authFacade.authenticateAndGetToken(optionalUser.get());
        } else {
            return register(provider, userInfo);
        }
    }

    public JwtResponse register(String provider, OAuth2UserInfo userInfo) {
        User user = new User();
        user.setProvider(AuthProvider.valueOf(provider.toUpperCase()));
        user.setEmail(userInfo.getEmail());
        user.setProfile(new UserProfile(userInfo.getName()));
        user.setGlobalRoles(Arrays.asList(new GlobalRole(UserGlobalRole.USER)));
        user.setVerified(true);
```

```
        return authFacade.authenticateAndGetToken(userService.save(user));
    }
}
```

```
package waterfall.flowfall.security.permission;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import waterfall.flowfall.model.enums.EntityType;
import waterfall.flowfall.model.enums.PermissionType;
import waterfall.flowfall.model.enums.UserGlobalRole;
import waterfall.flowfall.model.User;
import waterfall.flowfall.repository.RolePermissionRepository;
import waterfall.flowfall.util.SecurityContextUtils;

@Component
public class Access {

    private RolePermissionRepository rolePermissionRepository;

    @Autowired
    public Access(RolePermissionRepository rolePermissionRepository) {
        this.rolePermissionRepository = rolePermissionRepository;
    }

    /**
     * Checks if the authenticated user has the specified permission to operate the
specified entity
     * using the id of the board. <br/>
     * <br/>
     * For instance, if a user has the B.OWNER role on the specific board then that
means that the user has
     * the permissions to CREATE, READ, UPDATE, DELETE such entities as BOARD,
COLUMN, ROW, MESSAGE <b>that are
     * related to the specific board</b>.
     *
     * @param entityType Entity - the entity that is checked if a user has a
permission to operate it
     * @param permissionType Permission - the permission that is checked if user has
got such
     * @param entityId Long - the id of the entity the is used to check if a user has
a permission to operate
     *                 the related to the entity entities
     * @return boolean - true if a user is allowed to operate an entity, otherwise
false
     */
    public boolean require(EntityType entityType, PermissionType permissionType, Long
entityId) {
        if (isAdmin()) {
            return true;
        }

        User user = SecurityContextUtils.getAuthenticatedUser();

        return rolePermissionRepository.hasAccess(user.getId(), entityId, entityType,
permissionType);
    }

    public boolean isAdmin() {
        User user = SecurityContextUtils.getAuthenticatedUser();
```

```
        return user.getGlobalRoles().stream()
                .anyMatch(globalRole ->
globalRole.getName().equals(UserGlobalRole.ADMIN));
    }

}
```

```
package waterfall.flowfall.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.BoardColumn;
import waterfall.flowfall.repository.BoardColumnRepository;
import waterfall.flowfall.service.BoardColumnService;

import java.util.Optional;

@Service
public class BoardColumnServiceImpl implements BoardColumnService {

    private BoardColumnRepository boardColumnRepository;

    @Autowired
    public BoardColumnServiceImpl(BoardColumnRepository boardColumnRepository) {
        this.boardColumnRepository = boardColumnRepository;
    }

    @Override
    public Iterable<BoardColumn> findAll() {
        return boardColumnRepository.findAll();
    }

    @Override
    public Iterable<BoardColumn> findAllByBoardId(Long boardId) {
        return boardColumnRepository.findAllByBoardId(boardId);
    }

    @Override
    public Optional<BoardColumn> findById(Long id) {
        return boardColumnRepository.findById(id);
    }

    @Override
    public BoardColumn update(BoardColumn boardColumn) {
        return boardColumnRepository.findById(boardColumn.getId())
                .map(storedBoardColumn -> {
                    storedBoardColumn.setIndex(boardColumn.getIndex());
                    storedBoardColumn.setName(boardColumn.getName());
                    storedBoardColumn.setRows(boardColumn.getRows());

                    if(boardColumn.getBoard() != null) {
                        storedBoardColumn.setBoard(boardColumn.getBoard());
                    }

                    return boardColumnRepository.save(storedBoardColumn);
                }).orElse(null);
    }

    @Override
    public BoardColumn save(BoardColumn boardColumn) {
        return boardColumnRepository.save(boardColumn);
```

```
    }

    @Override
    public void delete(BoardColumn boardColumn) {
        boardColumnRepository.delete(boardColumn);
    }
}
```

```
package waterfall.flowfall.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.Board;
import waterfall.flowfall.model.enums.RoleType;
import waterfall.flowfall.repository.BoardRepository;
import waterfall.flowfall.service.BoardService;
import waterfall.flowfall.service.UserRoleService;

import java.util.Optional;

@Service
public class BoardServiceImpl implements BoardService {

    private BoardRepository boardRepository;
    private UserRoleService userRoleService;

    @Autowired
    public BoardServiceImpl(BoardRepository boardRepository, UserRoleService
userRoleService) {
        this.boardRepository = boardRepository;
        this.userRoleService = userRoleService;
    }

    @Override
    public Iterable<Board> findAll() {
        return boardRepository.findAll();
    }

    @Override
    public Optional<Board> findById(Long id) {
        return boardRepository.findById(id);
    }

    @Override
    public Board save(Board board) {
        Board savedBoard = boardRepository.save(board);
        userRoleService.addRole(savedBoard.getId(), board.getUser().getId(),
RoleType.BOARD_OWNER);
        return savedBoard;
    }

    @Override
    public Board update(Board board) {
        return boardRepository.findById(board.getId())
                .map(storedBoard -> {
                    storedBoard.setName(board.getName());
                    storedBoard.setBoardColumns(board.getBoardColumns());
                    storedBoard.setCollaborators(board.getCollaborators());
                    boardRepository.save(storedBoard);

                    return storedBoard;
```

```
                }).orElse(null);
    }

    @Override
    public void delete(Board board) {
        userRoleService.deleteRole(board.getId());
        boardRepository.delete(board);
    }

    @Override
    public Iterable<Board> findAllByUserId(Long id) {
        return boardRepository.findAllByUserId(id);
    }

    @Override
    public Iterable<Board> findAllCollabBoardsByUserId(Long id) {
        return boardRepository.findAllCollabBoardsByUserId(id);
    }
}
```

```
package waterfall.flowfall.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.RowMessage;
import waterfall.flowfall.repository.RowMessageRepository;
import waterfall.flowfall.service.RowMessageService;

import java.util.Date;
import java.util.Optional;

@Service
public class RowMessageServiceImpl implements RowMessageService {

    private RowMessageRepository rowMessageRepository;

    @Autowired
    public RowMessageServiceImpl(RowMessageRepository rowMessageRepository) {
        this.rowMessageRepository = rowMessageRepository;
    }

    @Override
    public Iterable<RowMessage> findAll() {
        return rowMessageRepository.findAll();
    }

    @Override
    public Optional<RowMessage> findById(Long id) {
        return rowMessageRepository.findById(id);
    }

    public Iterable<RowMessage> findAllByRowIdOrderByCreatedDesc(Long rowId) {
        return rowMessageRepository.findAllByRowIdOrderByCreatedDesc(rowId);
    }

    @Override
    public RowMessage update(RowMessage rowMessage) {
        return rowMessageRepository.save(rowMessage);
    }

    @Override
```

```
    public RowMessage save(RowMessage rowMessage) {
        rowMessage.setCreated(new Date());
        return rowMessageRepository.save(rowMessage);
    }

    @Override
    public void delete(RowMessage rowMessage) {
        rowMessageRepository.delete(rowMessage);
    }
}
package waterfall.flowfall.service.impl;

import org.springframework.stereotype.Service;
import waterfall.flowfall.model.Row;
import waterfall.flowfall.repository.RowRepository;
import waterfall.flowfall.service.RowService;

import java.util.Optional;

@Service
public class RowServiceImpl implements RowService {

    private RowRepository rowRepository;

    public RowServiceImpl(RowRepository rowRepository) {
        this.rowRepository = rowRepository;
    }

    @Override
    public Iterable<Row> findAll() {
        return rowRepository.findAll();
    }

    @Override
    public Iterable<Row> findAllByBoardColumnId(Long boardColumnId) {
        return rowRepository.findAllByBoardColumnId(boardColumnId);
    }

    @Override
    public Optional<Row> findById(Long id) {
        return rowRepository.findById(id);
    }

    @Override
    public Row update(Row row) {
        return rowRepository.findById(row.getId())
                .map(storedRow -> {
                    storedRow.setIndex(row.getIndex());
                    storedRow.setContent(row.getContent());

                    return rowRepository.save(storedRow);
                }).orElse(null);
    }

    @Override
    public Row save(Row row) {
        return rowRepository.save(row);
    }

    @Override
    public void delete(Row row) {
```

```
            rowRepository.delete(row);
        }
    }
```

```
package waterfall.flowfall.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.enums.RoleType;
import waterfall.flowfall.repository.UserRoleRepository;
import waterfall.flowfall.service.UserRoleService;


@Service
public class UserRoleServiceImpl implements UserRoleService {

    private UserRoleRepository userRoleRepository;

    @Autowired
    public UserRoleServiceImpl(UserRoleRepository userRoleRepository) {
        this.userRoleRepository = userRoleRepository;
    }

    @Override
    public void addRole(long entityId, long userId, long roleId) {
        userRoleRepository.addRole(entityId, userId, roleId);
    }

    @Override
    public void addRole(long entityId, long userId, RoleType roleType) {
        userRoleRepository.addRole(entityId, userId, roleType);
    }

    @Override
    public void deleteRole(long entityId, long userId, long roleId) {
        userRoleRepository.deleteRole(entityId, userId, roleId);
    }

    @Override
    public void deleteRole(long entityId, long userId) {
        userRoleRepository.deleteRole(entityId, userId);
    }

    @Override
    public void deleteRole(long entityId) {
        userRoleRepository.deleteRole(entityId);
    }
}
```

```
package waterfall.flowfall.service.impl;

import org.springframework.beans.BeanUtils;
import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.security.crypto.bcrypt.BCryptPasswordEncoder;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.User;
import waterfall.flowfall.repository.UserRepository;
import waterfall.flowfall.service.UserService;

import java.util.Optional;
```

```java
@Service
public class UserServiceImpl implements UserService {

    private UserRepository userRepository;
    private BCryptPasswordEncoder bcrypt;

    @Autowired
    public UserServiceImpl(UserRepository userRepository, BCryptPasswordEncoder
bcrypt) {
        this.userRepository = userRepository;
        this.bcrypt = bcrypt;
    }

    @Override
    public Iterable<User> findAll() {
        return userRepository.findAll();
    }

    @Override
    public Iterable<User> findCollaboratorsByBoardId(Long boardId) {
        return userRepository.findCollaboratorsByBoardId(boardId);
    }

    @Override
    public Optional<User> findById(Long id) {
        return userRepository.findById(id);
    }

    @Override
    public Optional<User> findByEmail(String email) {
        return userRepository.findByEmail(email);
    }

    @Override
    public User save(User user) {
        user.setPassword(bcrypt.encode(user.getPassword()));
        return userRepository.save(user);
    }

    @Override
    public User update(User user) {
        return userRepository.findById(user.getId())
                .map(savedUser -> {
                    BeanUtils.copyProperties(user, savedUser, "id");
                    return userRepository.save(savedUser);
                })
                .orElse(null);
    }

    @Override
    public void delete(User user) {
        userRepository.delete(user);
    }

    @Override
    public boolean existsByEmail(String email) {
        return userRepository.existsByEmail(email);
    }
}
```

```java
package waterfall.flowfall.service.impl;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Service;
import waterfall.flowfall.model.VerificationToken;
import waterfall.flowfall.repository.VerificationTokenRepository;
import waterfall.flowfall.service.VerificationTokenService;

import java.util.Date;
import java.util.Optional;

@Service
public class VerificationTokenServiceImpl implements VerificationTokenService {

    private VerificationTokenRepository verificationTokenRepository;

    @Autowired
    public VerificationTokenServiceImpl(VerificationTokenRepository
verificationTokenRepository) {
        this.verificationTokenRepository = verificationTokenRepository;
    }

    @Override
    public Optional<VerificationToken> findById(Long id) {
        return verificationTokenRepository.findById(id);
    }

    @Override
    public Optional<VerificationToken> findByToken(String token) {
        return verificationTokenRepository.findByToken(token);
    }

    @Override
    public VerificationToken save(VerificationToken token) {
        return verificationTokenRepository.save(token);
    }

    @Override
    public VerificationToken update(VerificationToken token) {
        return verificationTokenRepository.save(token);
    }

    @Override
    public void delete(VerificationToken token) {
        verificationTokenRepository.delete(token);
    }

    @Override
    public void deleteAllByExpirationDateBefore(Date date) {
        verificationTokenRepository.deleteAllByExpirationDateBefore(date);
    }
}

package waterfall.flowfall.validation;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.stereotype.Component;
import waterfall.flowfall.service.UserService;
import waterfall.flowfall.validation.annotations.VacantEmail;

import javax.validation.ConstraintValidator;
```

```
import javax.validation.ConstraintValidatorContext;

@Component
public class VacantEmailValidator implements ConstraintValidator<VacantEmail, String>
{

    private UserService userService;

    @Autowired
    public VacantEmailValidator(UserService userService) {
        this.userService = userService;
    }

    @Override
    public boolean isValid(String email, ConstraintValidatorContext context) {
        return !userService.existsByEmail(email);
    }
}
```

```
package waterfall.flowfall.websocket.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.Message;
import org.springframework.messaging.MessageChannel;
import org.springframework.messaging.simp.stomp.StompCommand;
import org.springframework.messaging.simp.stomp.StompHeaderAccessor;
import org.springframework.messaging.support.ChannelInterceptor;
import org.springframework.messaging.support.MessageHeaderAccessor;
import org.springframework.security.access.AccessDeniedException;
import org.springframework.stereotype.Component;
import waterfall.flowfall.security.jwt.JwtProvider;
import java.util.Objects;

@Component
public class WebSocketAuthInterceptor implements ChannelInterceptor {

    @Autowired
    private JwtProvider jwtProvider;

    @Override
    public Message<?> preSend(Message<?> message, MessageChannel channel) {
        final StompHeaderAccessor accessor =
MessageHeaderAccessor.getAccessor(message, StompHeaderAccessor.class);

        if (accessor != null && Objects.equals(accessor.getCommand(),
StompCommand.CONNECT)) {
            String token =
jwtProvider.getJwtFromHeader(accessor.getFirstNativeHeader(jwtProvider.JWT_HEADER_NAM
E));

            if (token == null || !jwtProvider.validateJwtToken(token)) {
                throw new AccessDeniedException("Cannot connect to websocket with
token: '" + token + "'");
            }
        }

        return message;
    }
}
```

```java
package waterfall.flowfall.websocket.configuration;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.context.annotation.Configuration;
import org.springframework.messaging.simp.config.ChannelRegistration;
import org.springframework.messaging.simp.config.MessageBrokerRegistry;
import org.springframework.web.socket.config.annotation.EnableWebSocketMessageBroker;
import org.springframework.web.socket.config.annotation.StompEndpointRegistry;
import
org.springframework.web.socket.config.annotation.WebSocketMessageBrokerConfigurer;

@Configuration
@EnableWebSocketMessageBroker
public class WebSocketConfiguration implements WebSocketMessageBrokerConfigurer {

    @Autowired
    private WebSocketAuthInterceptor webSocketAuthInterceptor;

    @Override
    public void registerStompEndpoints(StompEndpointRegistry registry) {
        registry.addEndpoint("/api/v1/webSocket")
                .setAllowedOrigins("*")
                .withSockJS();
    }

    @Override
    public void configureMessageBroker(MessageBrokerRegistry registry) {
        registry.setApplicationDestinationPrefixes("/message")
                .enableSimpleBroker("/message");
    }

    @Override
    public void configureClientInboundChannel(ChannelRegistration registration) {
        registration.interceptors(webSocketAuthInterceptor);
    }
}

package waterfall.flowfall.websocket.controller;

import org.springframework.beans.factory.annotation.Autowired;
import org.springframework.messaging.handler.annotation.DestinationVariable;
import org.springframework.messaging.handler.annotation.MessageMapping;
import org.springframework.messaging.handler.annotation.Payload;
import org.springframework.messaging.handler.annotation.SendTo;
import org.springframework.web.bind.annotation.CrossOrigin;
import org.springframework.web.bind.annotation.RestController;
import waterfall.flowfall.model.RowMessage;
import waterfall.flowfall.service.RowMessageService;
import waterfall.flowfall.service.UserService;
import waterfall.flowfall.websocket.WebSocketMessage;

@RestController
@CrossOrigin("*")
public class RowMessageSocketController {

    private RowMessageService rowMessageService;
    private UserService userService;

    @Autowired
    public RowMessageSocketController(RowMessageService rowMessageService,
UserService userService) {
```

```
        this.rowMessageService = rowMessageService;
        this.userService = userService;
    }

    @MessageMapping("/send/{rowId}")
    @SendTo("/message/{rowId}")
    public WebSocketMessage<RowMessage> sendMessage(@Payload
WebSocketMessage<RowMessage> message, @DestinationVariable Long rowId) {
        rowMessageService.save(message.getMessage());

        userService.findById(message.getMessage().getSender().getId())
                .ifPresent(sender -> message.getMessage().setSender(sender));

        return message;
    }

    @MessageMapping("/delete/{rowId}")
    @SendTo("/message/{rowId}")
    public WebSocketMessage<RowMessage> deleteMessage(@Payload
WebSocketMessage<RowMessage> message, @DestinationVariable Long rowId) {
        rowMessageService.delete(message.getMessage());
        return message;
    }
}
```

```
package waterfall.flowfall;

import org.springframework.boot.SpringApplication;
import org.springframework.boot.autoconfigure.SpringBootApplication;
import org.springframework.scheduling.annotation.EnableAsync;

@SpringBootApplication
@EnableAsync
public class FlowfallApplication {

   public static void main(String[] args) {
      SpringApplication.run(FlowfallApplication.class, args);
   }

}
```

Клієнтська частина

```
import { NgModule } from '@angular/core';
import { Routes, RouterModule } from '@angular/router';
import {LoginComponent} from './components/login/login.component';
import {BoardspaceComponent} from './components/boardspace/boardspace.component';
import {AuthGuard} from './auth/guards/auth.guard';
import {LoginGuard} from './auth/guards/login.guard';
import {BoardComponent} from './components/board/board.component';
import {Oauth2Component} from './components/oauth2/oauth2.component';
import {VerifyComponent} from './components/verify/verify.component';


const routes: Routes = [
  {path: '', pathMatch: 'full', redirectTo: 'boardspace'},
  {path: 'login', component: LoginComponent, canActivate: [LoginGuard]},
  {path: 'oauth2', component: Oauth2Component},
  {path: 'verify', component: VerifyComponent},
  {path: 'boardspace', component: BoardspaceComponent, canActivate: [AuthGuard]},
```

```
  {path: 'b/:id', component: BoardComponent, canActivate: [AuthGuard]}
];

@NgModule({
  imports: [RouterModule.forRoot(routes)],
  exports: [RouterModule]
})
export class AppRoutingModule { }

import { BrowserModule } from '@angular/platform-browser';
import { NgModule } from '@angular/core';

import { AppRoutingModule } from './app-routing.module';
import { AppComponent } from './app.component';
import { BrowserAnimationsModule } from '@angular/platform-browser/animations';
import { LoginComponent } from './components/login/login.component';
import { BoardspaceComponent } from './components/boardspace/boardspace.component';
import { BoardComponent } from './components/board/board.component';
import { HeaderComponent } from './components/header/header.component';
import {MaterialModule} from './modules/material/material.module';
import {TokenStorageService} from './auth/services/token-storage.service';
import {AuthService} from './auth/services/auth.service';
import {HTTP_INTERCEPTORS, HttpClientModule} from '@angular/common/http';
import {AuthInterceptor} from './auth/interceptors/auth-interceptor';
import {ErrorInterceptor} from './auth/interceptors/error-interceptor';
import {LoginGuard} from './auth/guards/login.guard';
import {AuthGuard} from './auth/guards/auth.guard';
import {FormsModule, ReactiveFormsModule} from '@angular/forms';
import {FlexLayoutModule} from '@angular/flex-layout';
import {BoardService} from './services/board.service';
import { AddRowDialogComponent } from './components/dialogs/add-row-dialog/add-row-
dialog.component';
import { AddColumnDialogComponent } from './components/dialogs/add-column-dialog/add-
column-dialog.component';
import {BoardColumnService} from './services/board-column.service';
import {RowService} from './services/row.service';
import { AddBoardDialogComponent } from './components/dialogs/add-board-dialog/add-
board-dialog.component';
import {UserService} from './services/user.service';
import { MenuDialogComponent } from './components/dialogs/menu-dialog/menu-
dialog.component';
import { Oauth2Component } from './components/oauth2/oauth2.component';
import { RowFeedDialogComponent } from './components/dialogs/row-feed-dialog/row-
feed-dialog.component';
import {WebsocketService} from './websocket/websocket.service';
import {RowMessageService} from './services/row-message.service';
import { ConfirmationDialogComponent } from './components/dialogs/confirmation-
dialog/confirmation-dialog.component';
import { MultiModeInputComponent } from './components/multi-mode-input/multi-mode-
input.component';
import {CollaboratorService} from './services/collaborator.service';
import {ResponseService} from './services/response.service';
import { VerifyComponent } from './components/verify/verify.component';

@NgModule({
  declarations: [
    AppComponent,
    LoginComponent,
    BoardspaceComponent,
    BoardComponent,
    HeaderComponent,
```

```
    AddRowDialogComponent,
    AddColumnDialogComponent,
    AddBoardDialogComponent,
    MenuDialogComponent,
    Oauth2Component,
    MultiModeInputComponent,
    RowFeedDialogComponent,
    ConfirmationDialogComponent,
    VerifyComponent
  ],
  imports: [
    MaterialModule,
    BrowserModule,
    AppRoutingModule,
    BrowserAnimationsModule,
    HttpClientModule,
    FormsModule,
    FlexLayoutModule,
    ReactiveFormsModule
  ],
  providers: [
    TokenStorageService,
    AuthService,
    WebsocketService,
    {provide: HTTP_INTERCEPTORS, useClass: AuthInterceptor, multi: true},
    {provide: HTTP_INTERCEPTORS, useClass: ErrorInterceptor, multi: true},
    AuthGuard,
    LoginGuard,
    BoardService,
    BoardColumnService,
    RowService,
    UserService,
    RowMessageService,
    CollaboratorService,
    ResponseService
  ],
  entryComponents: [
    ConfirmationDialogComponent,
    AddRowDialogComponent,
    AddColumnDialogComponent,
    AddBoardDialogComponent,
    MenuDialogComponent,
    RowFeedDialogComponent
  ],
  bootstrap: [AppComponent]
})
export class AppModule { }

import { Component } from '@angular/core';

@Component({
  selector: 'app-root',
  templateUrl: './app.component.html',
  styleUrls: ['./app.component.scss']
})
export class AppComponent {
  title = 'flowfall-frontend';
}

<div style="height:100%" fxLayout="column">
    <app-header></app-header>
```

```
    <div fxFlex="1 0">
      <router-outlet></router-outlet>
    </div>
</div>

import { Injectable } from '@angular/core';
import {CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router}
from '@angular/router';
import { Observable } from 'rxjs';
import {AuthService} from '../services/auth.service';

const REDIRECT_PAGE = 'login';

@Injectable()
export class AuthGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean |
UrlTree> | boolean | UrlTree {

    if (!this.authService.isAuthenticated()) {
      this.router.navigate([REDIRECT_PAGE]);
      return false;
    }

    return true;
  }

}
import { Injectable } from '@angular/core';
import {CanActivate, ActivatedRouteSnapshot, RouterStateSnapshot, UrlTree, Router}
from '@angular/router';
import { Observable } from 'rxjs';
import {AuthService} from '../services/auth.service';

const REDIRECT_PAGE = 'boardspace';

@Injectable()
export class LoginGuard implements CanActivate {

  constructor(private authService: AuthService, private router: Router) { }

  canActivate(
    next: ActivatedRouteSnapshot,
    state: RouterStateSnapshot): Observable<boolean | UrlTree> | Promise<boolean |
UrlTree> | boolean | UrlTree {

    if (this.authService.isAuthenticated()) {
      this.router.navigate([REDIRECT_PAGE]);
      return false;
    }

    return true;
  }

}
```

```
import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from
'@angular/common/http';
import {Observable} from 'rxjs';
import {TokenStorageService} from '../services/token-storage.service';
import {Injectable} from '@angular/core';

@Injectable()
export class AuthInterceptor implements HttpInterceptor {

  constructor(private tokenStorage: TokenStorageService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    let authRequest = req;
    let token = this.tokenStorage.getToken();

    if (token != null) {
      authRequest = req.clone({headers: req.headers.set('Authorization', 'Bearer ' +
token)});
    }

    return next.handle(authRequest);
  }

}

import {HttpEvent, HttpHandler, HttpInterceptor, HttpRequest} from
'@angular/common/http';
import {Observable, throwError} from 'rxjs';
import {TokenStorageService} from '../services/token-storage.service';
import {catchError} from 'rxjs/operators';
import {Injectable} from '@angular/core';

@Injectable()
export class ErrorInterceptor implements HttpInterceptor {

  constructor(private tokenStorage: TokenStorageService) { }

  intercept(req: HttpRequest<any>, next: HttpHandler): Observable<HttpEvent<any>> {
    return next.handle(req).pipe(catchError(err => {
      if ([401, 403].indexOf(err.status) > 0) {
        this.tokenStorage.clear();
        window.location.reload();
      }

      return throwError(err);
    }));
  }
}


import { Injectable } from '@angular/core';
import {HttpClient} from '@angular/common/http';
import {User} from '../../models/User';
import {environment} from '../../../environments/environment';
import {Observable} from 'rxjs';
import {JwtResponse} from '../JwtResponse';
import {TokenStorageService} from './token-storage.service';
import {RegisterRequest} from '../../models/requests/RegisterRequest';
import {LoginRequest} from '../../models/requests/LoginRequest';

@Injectable()
```

```
export class AuthService {

  private BASE_URL = `${environment.api_url}/auth`;

  constructor(private http: HttpClient, private tokenStorage: TokenStorageService) {
}

  authenticate(loginRequest: LoginRequest): Observable<JwtResponse> {
    let url = `${this.BASE_URL}/login`;

    return this.http.post<JwtResponse>(url, loginRequest);
  }

  register(registerRequest: RegisterRequest): Observable<JwtResponse> {
    let url = `${this.BASE_URL}/register`;

    return this.http.post<JwtResponse>(url, registerRequest);
  }


  logout() {
    this.tokenStorage.clear();
  }

  isAuthenticated(): boolean {
    let token = this.tokenStorage.getToken();

    return this.tokenStorage.isTokenValid(token);
  }
}

import { Injectable } from '@angular/core';
import {JwtHelperService} from '@auth0/angular-jwt';
import {JwtResponse} from '../JwtResponse';

const TOKEN_KEY = 'AuthAccessToken';
const EMAIL_KEY = 'AuthEmail';

@Injectable()
export class TokenStorageService {

  private jwtService = new JwtHelperService();

  constructor() { }

  clear() {
    window.localStorage.clear();
  }

  saveData(data: JwtResponse) {
    this.saveToken(data.accessToken);
    this.saveEmail(data.email);
  }

  isTokenValid(token: string) {
    let isTokenValid = false;

    if(token != null) {
      try {
        isTokenValid = !this.jwtService.isTokenExpired(token);
      } catch {
```

```
        this.clear();
      }
    }

    return isTokenValid;
  }

  saveToken(token: string) {
    window.localStorage.removeItem(TOKEN_KEY);
    window.localStorage.setItem(TOKEN_KEY, token);
  }

  getToken(): string {
    return window.localStorage.getItem(TOKEN_KEY);
  }

  saveEmail(email: string) {
    window.localStorage.removeItem(EMAIL_KEY);
    window.localStorage.setItem(EMAIL_KEY, email);
  }

  getEmail(): string {
    return window.localStorage.getItem(EMAIL_KEY);
  }

  getId() {
    return this.decodeJwt(this.getToken()).id;
  }

  private decodeJwt(jwt: string) {
    const jwtData = jwt.split('.')[1];
    const decodedJwt = JSON.parse(atob(jwtData));

    return decodedJwt;
  }

}

import {Component, OnInit} from '@angular/core';
import {ActivatedRoute, Router} from '@angular/router';
import {Board} from '../../models/Board';
import {BoardService} from '../../services/board.service';
import {CdkDragDrop, moveItemInArray, transferArrayItem} from '@angular/cdk/drag-
drop';
import {BoardColumn} from '../../models/BoardColumn';
import {Row} from '../../models/Row';
import {MatDialog} from '@angular/material';
import {AddRowDialogComponent} from '../dialogs/add-row-dialog/add-row-
dialog.component';
import {AddColumnDialogComponent} from '../dialogs/add-column-dialog/add-column-
dialog.component';
import {BoardColumnService} from '../../services/board-column.service';
import {RowService} from '../../services/row.service';
import {User} from '../../models/User';
import {UserService} from '../../services/user.service';
import {MenuDialogComponent} from '../dialogs/menu-dialog/menu-dialog.component';
import {TokenStorageService} from '../../auth/services/token-storage.service';
import {RowFeedDialogComponent} from '../dialogs/row-feed-dialog/row-feed-
dialog.component';
import {CollaboratorService} from '../../services/collaborator.service';
```

```
enum FieldMode {
  EDIT = 'edit', VIEW = 'view'
}

@Component({
  selector: 'app-board',
  templateUrl: './board.component.html',
  styleUrls: ['./board.component.scss']
})
export class BoardComponent implements OnInit {
  isOwner: boolean = false;

  boardId: string;
  currentBoard: Board = new Board();
  connectedList: string[] = [];
  collaborators: User[] = [];

  isMouseDown = false;
  clickX;
  clickScrollLeft;

  menuRef;
  isMenuOpened = false;

  constructor(private route: ActivatedRoute, private router: Router, private dialog:
MatDialog,
              private boardService: BoardService, private boardColumnService:
BoardColumnService, private rowService: RowService,
              private userService: UserService, private collaboratorService:
CollaboratorService,
              private tokenStorage: TokenStorageService) {
    this.route.params.subscribe(
      params => {
        this.boardId = params['id'];

        this.boardService.getBoardById(this.boardId).subscribe(
          data => {
            this.currentBoard = data;
            this.fillConnectedList(data.boardColumns);


this.collaboratorService.getCollaboratorsByBoardId(this.currentBoard.id).subscribe(
              collabs => this.collaborators = collabs,
              error => console.log(error)
            );


this.collaboratorService.getOwnerByBoardId(this.currentBoard.id).subscribe(
              owner => this.isOwner = owner.id.toString() === tokenStorage.getId()
            );
          },
          error => console.log(error)
        );

      }
    );
  }

  ngOnInit() {
  }
```

```
  dropRow(event: CdkDragDrop<Row[]>) {
    if (event.previousContainer === event.container) {
      if (event.previousIndex === event.currentIndex)
        return;

      moveItemInArray(event.container.data, event.previousIndex, event.currentIndex);
    } else {
      transferArrayItem(event.previousContainer.data,
        event.container.data,
        event.previousIndex,
        event.currentIndex);
    }

    this.recalculateIndices(this.currentBoard);
    this.boardService.updateBoard(this.currentBoard).subscribe();
  }

  dropColumn(event: CdkDragDrop<BoardColumn[]>) {
    if (event.previousIndex === event.currentIndex)
      return;

    moveItemInArray(this.currentBoard.boardColumns, event.previousIndex,
event.currentIndex);

    this.recalculateIndices(this.currentBoard);
    this.boardService.updateBoard(this.currentBoard).subscribe();
  }

  fillConnectedList(columns: BoardColumn[]) {
    columns.forEach(data => {
      this.connectedList.push(data.name);
    });
  }

  recalculateIndices(board: Board) {
    for (let i = 0; i < board.boardColumns.length; i++) {
      board.boardColumns[i].index = i + 1;
      for (let j = 0; j < board.boardColumns[i].rows.length; j++) {
        board.boardColumns[i].rows[j].index = j + 1;
      }
    }
  }

  addColumn() {
    const dialogRef = this.dialog.open(AddColumnDialogComponent, {
      width: '250px'
    });

    dialogRef.afterClosed().subscribe(data => {
      if (data !== undefined) {
        this.boardColumnService.addBoardColumn(this.currentBoard.id,
          new BoardColumn(data.name, this.currentBoard.boardColumns.length + 1,
this.currentBoard.id))
          .subscribe(
          column => {
            this.currentBoard.boardColumns.push(column);
            this.connectedList.push(column.name);
          } ,
          error => console.log(error)
        );
      }
```

```
    });
  }

  addRow(column: BoardColumn) {
    const dialogRef = this.dialog.open(AddRowDialogComponent, {
      width: '250px'
    });

    dialogRef.afterClosed().subscribe(data => {
      if (data !== undefined) {
        this.rowService.addRow(this.currentBoard.id, column.id,
          new Row(data.name, column.rows.length + 1, column.id)
        ).subscribe(
          row => column.rows.push(row),
          error => console.log(error)
        );
      }
    });
  }

  openMenu() {
    if (this.menuRef !== undefined && this.isMenuOpened === true) {
      this.menuRef.close();
    } else {
      this.menuRef = this.dialog.open(MenuDialogComponent, {
        data: this.currentBoard.id,
        disableClose: true,
        hasBackdrop: false,
        width: '400px',
        height: '80vh',
        position: {
          top: '100px',
          right: '20px'
        },
      });

      this.menuRef.afterOpened().subscribe(() => this.isMenuOpened = true);
      this.menuRef.afterClosed().subscribe(() => this.isMenuOpened = false);
    }
  }

  openRowFeed(row: Row, column: BoardColumn) {
    let dialogRef = this.dialog.open(RowFeedDialogComponent, {
      data: {row: row, colId: column.id, boardId: this.currentBoard.id},
      width: '50vw',
      height: '90vh'
    });
  }

  deleteRow(column: BoardColumn, row: Row) {
    this.rowService.deleteRow(this.currentBoard.id, column.id, row.id).subscribe(
      () => column.rows = column.rows.filter(colRow => colRow !== row),
      error => console.log(error)
    );
  }

  deleteColumn(column: BoardColumn) {
    this.boardColumnService.deleteBoardColumn(this.currentBoard.id,
column.id).subscribe(
      () => this.currentBoard.boardColumns =
this.currentBoard.boardColumns.filter(col => col !== column),
```

```
      error => console.log(error)
    );
  }

  deleteBoard() {
    this.boardService.deleteBoard(this.currentBoard.id).subscribe(
      () => this.router.navigate(['/boardspace']),
      error => console.log(error)
    );
  }

  editBoardName(value) {
    if (value !== this.currentBoard.name) {
      let oldName = this.currentBoard.name;
      this.currentBoard.name = value;

      this.boardService.updateBoard(this.currentBoard).subscribe(
        () => {},
        () => this.currentBoard.name = oldName
      );
    }
  }

  editColumnName(value, column: BoardColumn) {
    if (value !== column.name) {
      let oldName = column.name;
      column.name = value;

      this.boardColumnService.updateBoardColumn(this.currentBoard.id,
column).subscribe(
        () => {},
        () => column.name = oldName
      );
    }
  }

  showDeleteIcon(event) {
    event.target.getElementsByClassName('delete-icon')[0].style.visibility =
'visible';
  }

  hideDeleteIcon(event) {
    event.target.getElementsByClassName('delete-icon')[0].style.visibility =
'hidden';
  }

  inviteCollab(event) {
    let collabEmail = event.value;
    this.collaboratorService.inviteCollaborator(this.currentBoard.id,
collabEmail).subscribe(
      data => console.log('Invited'),
      error => console.log('Could not invite')
    );

  }

  scrollBoardContent(event) {
    let element = event.target;
    element.scrollLeft = this.clickScrollLeft + (this.clickX - event.pageX);
  }
```

```
  mouseDown(event) {
    this.isMouseDown = true;
    this.clickX = event.pageX;
    this.clickScrollLeft = event.target.scrollLeft;
  }

  mouseUp() {
    this.isMouseDown = false;
  }
}
```

```
<div class="board-container" fxLayout="column">
  <div class="board-title"  fxLayout="row" fxLayoutAlign="start center"
fxLayoutGap="10px">
    <div class="board-name"
         (mouseenter)="isOwner && showDeleteIcon($event)" (mouseleave)="isOwner &&
hideDeleteIcon($event)"
          fxLayout="row" fxLayoutAlign="start center">
      <multi-mode-input [value]="currentBoard.name" [minWidthStyle]="'220px'"
[allowEmpty]="false"
                        (valueChanged)="editBoardName($event)">
      </multi-mode-input>
      <div class="material-icons delete-icon" *ngIf="isOwner"
(click)="deleteBoard()">delete</div>
    </div>

    <div fxLayout="row" fxLayoutAlign="start center" fxLayoutGap="10px" fxHide.lt-md>
      <button mat-raised-button color="primary"
(click)="inviteCollab(emailInput)">Invite</button>

      <mat-form-field class="form-field-input">
        <input #emailInput class="title-input" matInput placeholder="Collaborator
email"/>
      </mat-form-field>
    </div>

    <div class="menu" (click)="openMenu()"
         fxLayout="row" fxLayoutAlign="start center" fxLayoutGap="5px">
      <div fxHide.lt-sm>Menu</div>
      <div class="material-icons">menu</div>
    </div>

  </div>

  <div class="board-content"
       (mousemove)="isMouseDown && scrollBoardContent($event)"
       (mousedown)="mouseDown($event)" (mouseup)="mouseUp()" (mouseleave)="mouseUp()"
       cdkDropList
       [cdkDropListData]="currentBoard.boardColumns"
       (cdkDropListDropped)="dropColumn($event)"
       cdkDropListOrientation="horizontal"
       fxFlex="1 0" fxLayout="row" fxLayoutGap="10px">

    <div class="column" *ngFor="let column of currentBoard.boardColumns" cdkDrag
         fxFlexAlign="start">
      <div class="column-title" (mouseenter)="showDeleteIcon($event)"
(mouseleave)="hideDeleteIcon($event)"
           fxLayout="row" fxLayoutAlign="space-between center">
        <multi-mode-input [value]="column.name" [allowEmpty]="false"
(valueChanged)="editColumnName($event, column)">
        </multi-mode-input>
```

```
        <div class="material-icons delete-icon" (click)="deleteColumn(column)"
             fxFlexAlign="center">
          delete
        </div>
      </div>

      <div [id]="column.name"
           cdkDropList
           [cdkDropListData]="column.rows"
           (cdkDropListDropped)="dropRow($event)"
           [cdkDropListConnectedTo]="connectedList"
           fxLayout="column" fxLayoutGap="5px">
        <div class="row" cdkDrag *ngFor="let row of column.rows"
(click)="openRowFeed(row, column)"
             (mouseenter)="showDeleteIcon($event)"
(mouseleave)="hideDeleteIcon($event)"
             fxLayout="row" fxLayoutAlign="space-between">
          <div class="row-content">{{row.name}}</div>
          <div class="material-icons delete-icon" (click)="deleteRow(column, row)"
               fxFlexAlign="center">
            delete
          </div>
        </div>

        <div class="add-row-box" (click)="addRow(column)">Add row</div>
      </div>
    </div>

    <div class="add-column" fxFlexAlign="start" fxLayoutAlign="start center"
(click)="addColumn()">
      <div>Add column</div>
    </div>

  </div>
</div>

import { Component, OnInit } from '@angular/core';
import {BoardService} from '../../services/board.service';
import {TokenStorageService} from '../../auth/services/token-storage.service';
import {Board} from '../../models/Board';
import {ActivatedRoute, Router} from '@angular/router';
import {MatDialog} from "@angular/material";
import {AddBoardDialogComponent} from "../dialogs/add-board-dialog/add-board-
dialog.component";
import {User} from "../../models/User";

@Component({
  selector: 'app-boardspace',
  templateUrl: './boardspace.component.html',
  styleUrls: ['./boardspace.component.scss']
})
export class BoardspaceComponent implements OnInit {

  boards: Board[] = [];
  collabBoards: Board[] = [];

  constructor(private boardService: BoardService, private tokenStorage:
TokenStorageService,
              private router: Router, private activatedRoute: ActivatedRoute, private
dialog: MatDialog) { }
```

```
  ngOnInit() {
    this.boardService.getBoards().subscribe(
      data => {
        this.boards = data;
      },
      error => console.log(error)
    );

    this.boardService.getCollaborativeBoards().subscribe(
      data => {
        this.collabBoards = data;
      },
      error => console.log(error)
    );
  }

  navigateToBoard(board: Board) {
    this.router.navigate([`/b`, board.id]);
  }

  addBoard() {
    let dialogRef = this.dialog.open(AddBoardDialogComponent,{
      width: '250px'
    });

    dialogRef.afterClosed().subscribe(data => {
      if (data !== undefined) {
        data.user = new User(parseInt(this.tokenStorage.getId()));

        this.boardService.addBoard(data).subscribe(board => {
          this.boards.push(board);
          this.router.navigate(['/b', board.id]);
        });
      }
    });
  }

}
```

```
<div class="boardspace-container">
  <p>Your boards:</p>

  <div fxLayout="row" fxLayoutGap="10px">
    <mat-card (click)="addBoard()">+</mat-card>
    <mat-card *ngFor="let board of boards" (click)="navigateToBoard(board)">
      {{board.name}}
    </mat-card>
  </div>

  <div *ngIf="collabBoards.length > 0">
    <p>Your collaborating boards:</p>

    <div fxLayout="row" fxLayoutGap="10px">
      <mat-card *ngFor="let board of collabBoards" (click)="navigateToBoard(board)">
        {{board.name}}
      </mat-card>
    </div>
  </div>

</div>
```

```
import { Component, OnInit } from '@angular/core';
import {AuthService} from '../../auth/services/auth.service';

@Component({
  selector: 'app-header',
  templateUrl: './header.component.html',
  styleUrls: ['./header.component.scss']
})
export class HeaderComponent implements OnInit {

  constructor(private authService: AuthService) { }

  ngOnInit() {
  }

  logout() {
    this.authService.logout();
    window.location.reload();
  }
}
```

```
<mat-toolbar *ngIf="authService.isAuthenticated()" color="primary"
             fxLayoutAlign="center center" fxLayout="row" fxLayoutGap="20px">
  <div class="title" routerLink="/boardspace">Flowfall</div>
  <div class="material-icons home" routerLink="/boardspace">home</div>

  <div class="material-icons person">person</div>
  <div class="material-icons logout" (click)="logout()">exit_to_app</div>
</mat-toolbar>
```

```
import { Component, OnInit } from '@angular/core';
import {AuthService} from '../../auth/services/auth.service';
import {TokenStorageService} from '../../auth/services/token-storage.service';
import {environment} from '../../../environments/environment';
import {FACEBOOK_PROVIDER, GOOGLE_PROVIDER, REDIRECT_URI} from
'../../constants/OAuth2Constants';
import {FormBuilder, FormGroup, Validators} from '@angular/forms';
import {RegisterRequest} from '../../models/requests/RegisterRequest';
import {PasswordMatch} from '../../validators/PasswordMatch';
import {LoginRequest} from '../../models/requests/LoginRequest';
import {ResponseService} from '../../services/response.service';


@Component({
  selector: 'app-login',
  templateUrl: './login.component.html',
  styleUrls: ['./login.component.scss']
})
export class LoginComponent implements OnInit {

  private oauth2ProviderOpenedTimer;

  private GOOGLE_PROVIDER_URL =
`${environment.oauth2_url}?provider=${GOOGLE_PROVIDER}&redirect_uri=${REDIRECT_URI}`;
  private FACEBOOK_PROVIDER_URL =
`${environment.oauth2_url}?provider=${FACEBOOK_PROVIDER}&redirect_uri=${REDIRECT_URI}
`;

  registerForm: FormGroup;
  loginForm: FormGroup;
  loginMode: boolean = true;
```

```
  constructor(private authService: AuthService, private tokenStorage:
TokenStorageService, private formBuilder: FormBuilder,
              private responseService: ResponseService) {
    this.initializeLoginForm();
    this.initializeRegisterForm();
  }

  ngOnInit() {

  }

  navigateToProvider(provider: string) {
    const width = 500;
    const height = 600;

    const left = (screen.width / 2) - (width / 2);
    const top = (screen.height / 2) - (height / 2);

    const win = window.open(provider, 'Provider authentication',
      `width=${width}, height=${height}, left=${left}, top=${top}`);

    clearInterval(this.oauth2ProviderOpenedTimer);
    this.oauth2ProviderOpenedTimer = setInterval(() => {
      if (win.closed) {
        clearInterval(this.oauth2ProviderOpenedTimer);
        location.reload();
      }
    }, 1000);
  }

  get loginCtrls() {
    return this.loginForm.controls;
  }

  get registerCtrls() {
    return this.registerForm.controls;
  }

  switchForms() {
    this.loginMode = !this.loginMode;
  }

  initializeLoginForm() {
    this.loginForm = this.formBuilder.group({
      email: ['', Validators.required],
      password: ['', Validators.required]
    });
  }

  initializeRegisterForm() {
    this.registerForm = this.formBuilder.group({
      fullname: ['', Validators.required],
      email: ['', [Validators.required, Validators.email]],
      password: ['', Validators.required],
      confirmPassword: ['', Validators.required]
    }, {
      validators: PasswordMatch('password', 'confirmPassword')
    });
  }
```

```
  login() {
    if (this.loginForm.invalid) {
      return;
    }

    const loginRequest: LoginRequest = this.loginForm.value;

    this.authService.authenticate(loginRequest).subscribe(
      data => {
        if (data.enabled) {
          this.tokenStorage.saveData(data);
          window.location.reload();
        }
      },
      error => this.responseService.handleMessage('Error occurred. Try again')
    );
  }

  register() {
    if (this.registerForm.invalid) {
      return;
    }

    const registerRequest: RegisterRequest = this.registerForm.value;
    registerRequest.redirectUri = environment.redirect_uri;

    this.authService.register(registerRequest).subscribe(
      () => {
        this.responseService.handleMessage('Confirmation is sent to the email');
        this.loginMode = true;
      },
        error => this.responseService.handleFieldErrors(error, this.registerForm));
  }
}
```

```
<div class="login-wrapper" fxLayout="row" fxLayoutAlign="center center">


  <div class="login-container" fxLayout="row" fxLayoutAlign="start center">

    <div class="left-block" fxLayout="column" fxLayoutAlign="center center">
      <div class="title-block" fxLayout="column" fxLayoutAlign="center center">
        <div class="login-title">FLOWFALL</div>

        <div>Task tracker</div>
        <div>Optimize your work and ideas</div>

        <div style="padding-top: 50px"></div>

        <div class="title-points"
             fxLayout="column" fxLayoutAlign="center center" fxLayoutGap="5px">
          <div>Create board</div>
          <div>Fill it with content</div>
          <div>Share it with collaborators</div>
        </div>
      </div>
    </div>

    <div class="right-block" fxLayout="column" fxLayoutAlign="center center"
fxLayoutGap="5px">
     <div class="auth-form" fxLayout="column" fxLayoutAlign="center center"
```

```
fxLayoutGap="5px">
       <div class="socials" fxLayout="column" fxLayoutAlign="center center"
fxLayoutGap="10px">
         <div class="social-btn"
              (click)="navigateToProvider(FACEBOOK_PROVIDER_URL)"
              fxLayout="row" fxLayoutAlign="start center" >
           <img src="/assets/logos/fb-logo.png" alt="Facebook logo" width="30px"
height="30px">
           <div class="social-btn-text">Sign in with Facebook</div>
         </div>

         <div class="social-btn"
              (click)="navigateToProvider(GOOGLE_PROVIDER_URL)"
              fxLayout="row" fxLayoutAlign="start center">
           <img src="/assets/logos/google-logo.png" alt="Google logo" width="30px"
height="30px">
           <div class="social-btn-text">Sign in with Google</div>
         </div>
       </div>

       <form *ngIf="loginMode" [formGroup]="loginForm" (ngSubmit)="login()"
             fxLayout="column" fxLayoutAlign="center center" fxLayoutGap="5px">
         <mat-form-field>
           <input matInput formControlName="email" placeholder="Email"/>
           <mat-error *ngIf="loginCtrls.email.errors?.required">Email is
required</mat-error>
         </mat-form-field>

         <mat-form-field>
           <input matInput formControlName="password" placeholder="Password"
type="password"/>
           <mat-error *ngIf="loginCtrls.password.errors?.required">Password is
required</mat-error>
         </mat-form-field>

         <button mat-raised-button type="submit" color="primary">Sign in</button>
         <div class="auth-text">
           Don't have an account? <span class="auth-btn" (click)="switchForms()">Sign
up</span>
         </div>
       </form>

       <form *ngIf="!loginMode" [formGroup]="registerForm" (ngSubmit)="register()"
             fxLayout="column" fxLayoutAlign="center center" fxLayoutGap="5px">
         <mat-form-field>
           <input matInput formControlName="fullname" placeholder="Full name"/>
           <mat-error *ngIf="registerCtrls.fullname.errors?.required">Full name is
required</mat-error>
         </mat-form-field>

         <mat-form-field>
           <input matInput formControlName="email" placeholder="Email"/>
           <mat-error *ngIf="registerCtrls.email.errors?.email">Enter a valid
email</mat-error>
           <mat-error *ngIf="registerCtrls.email.errors?.required ">Email is
required</mat-error>
           <mat-error
*ngIf="registerCtrls.email.errors?.serverError">{{registerCtrls.email.errors?.serverE
rror}}</mat-error>
         </mat-form-field>
```

```
          <mat-form-field>
            <input matInput formControlName="password" placeholder="Password"
type="password"/>
            <mat-error *ngIf="registerCtrls.password.errors?.required">Password is
required</mat-error>
          </mat-form-field>

          <mat-form-field>
            <input matInput formControlName="confirmPassword" placeholder="Confirm
password" type="password"/>
            <mat-error *ngIf="registerCtrls.confirmPassword.errors?.required">Password
is required</mat-error>
            <mat-error
*ngIf="registerCtrls.confirmPassword.errors?.passwordMatch">Password should
match</mat-error>
          </mat-form-field>

          <button mat-raised-button type="submit" color="primary">Sign up</button>

          <div class="auth-text">
            Have an account? <span class="auth-btn" (click)="switchForms()">Sign
in</span>
          </div>
        </form>

      </div>
    </div>

  </div>
</div>

import {Component, Inject, OnInit} from '@angular/core';
import {Row} from '../../../models/Row';
import {MAT_DIALOG_DATA, MatDialog, MatDialogRef} from '@angular/material';
import {RowService} from '../../../services/row.service';

import {RowMessage} from '../../../models/RowMessage';
import {WebSocketRowMessage} from '../../../websocket/WebSocketRowMessage';
import {TokenStorageService} from '../../../auth/services/token-storage.service';
import {User} from '../../../models/User';
import {WebsocketService} from '../../../websocket/websocket.service';
import {RowMessageService} from '../../../services/row-message.service';
import {ConfirmationDialogComponent} from '../confirmation-dialog/confirmation-
dialog.component';

@Component({
  selector: 'app-row-feed-dialog',
  templateUrl: './row-feed-dialog.component.html',
  styleUrls: ['./row-feed-dialog.component.scss']
})
export class RowFeedDialogComponent implements OnInit {

  row: Row = new Row();

  messages: RowMessage[] = [];

  constructor(public dialogRef: MatDialogRef<RowFeedDialogComponent>,
@Inject(MAT_DIALOG_DATA) public data: {row: Row, colId: number, boardId: number},
              private rowService: RowService, private tokenStorage:
TokenStorageService,
              private websocketService: WebsocketService, private rowMessageService:
```

```
RowMessageService,
              private dialog: MatDialog) {
    this.row = data.row;

    this.rowMessageService.getRowMessagesByRowId(this.data.boardId, this.data.colId,
this.row.id).subscribe(
      rowMessages => this.messages = rowMessages
    );

    this.websocketService.initWebSocketConnection(this.row.id, (message) => {
      this.onWebSocketMessageReceived(message);
    });
  }

  ngOnInit() {
  }

  save() {
    this.dialogRef.close(this.row);
  }

  close() {
    this.dialogRef.close();
  }

  editRowName(value) {
    let oldName = this.row.name;

    if (value !== oldName) {
      this.row.name = value;

      this.rowService.updateRow(this.data.boardId, this.data.colId,
this.row).subscribe(
        () => {},
        error => this.row.name = oldName
      );
    }
  }

  editRowContent(value) {
    let oldContent = this.row.content;

    if (value !== oldContent) {
      this.row.content = value;

      this.rowService.updateRow(this.data.boardId, this.data.colId,
this.row).subscribe(
        () => {},
        error => this.row.content = oldContent
      );
    }
  }

  onWebSocketMessageReceived(message: WebSocketRowMessage) {
    if (message.type === 'SEND') {
      this.messages.unshift(message.message);
    } else if (message.type === 'DELETE') {
      this.messages = this.messages.filter(msg => msg.id !== message.message.id);
    }
  }
```

```
  addComment(messageInput) {
    this.websocketService.sendMessage(this.row.id,
this.createWebSocketMessage(messageInput.value));
    messageInput.value = '';
  }

  deleteComment(message: RowMessage) {
    let confirmRef = this.dialog.open(ConfirmationDialogComponent, {
      data: 'Are you sure you want to delete comment?'
    });

    confirmRef.afterClosed().subscribe(confirm => {
      if (confirm) {
        let comment = document.getElementById(String(message.id));

        comment.classList.add('animated', 'bounceOut', 'fast');
        comment.addEventListener('animationend', () => {
          this.websocketService.deleteMessage(this.row.id, new
WebSocketRowMessage('DELETE', message));
          this.messages = this.messages.filter(msg => msg.id !== message.id);
        });
      }
    });
  }

  editComment(value, msg: RowMessage) {
    let oldComment = msg.text;

    if (value !== oldComment) {
      msg.text = value;

      this.rowMessageService.update(this.data.boardId, this.data.colId, this.row.id,
msg).subscribe(
        () => {},
        error => msg.text = oldComment
      );
    }
  }

  createWebSocketMessage(message: string): WebSocketRowMessage {
    let rowMessage = new RowMessage();
    rowMessage.row = this.row;
    rowMessage.sender = new User(parseInt(this.tokenStorage.getId()));
    rowMessage.text = message;

    return new WebSocketRowMessage('SEND', rowMessage);
  }

  isCommentOwner(msg: RowMessage) {
    return msg.sender.id == this.tokenStorage.getId();
  }
}
```

```
<div class="rowfeed-wrapper" fxLayout="column" fxLayoutGap="20px">
  <div class="rowfeed-name" fxLayoutAlign="start center" fxLayoutGap="5px">
    <div class="material-icons">title</div>
    <multi-mode-input class="bold" style="width: 100%" [value]="row.name"
[minWidthStyle]="'100%'" [allowEmpty]="false"
                      (valueChanged)="editRowName($event)">
    </multi-mode-input>
  </div>
```

```
  <div class="rowfeed-content">
    <div fxLayout="row" fxLayoutAlign="start center" fxLayoutGap="5px">
      <div class="material-icons">description</div>
      <div class="bold">Row content</div>
    </div>

    <div class="content-field">
        <multi-mode-input [value]="row.content" [minWidthStyle]="'100%'"
[textArea]="true"
                          (valueChanged)="editRowContent($event)">
        </multi-mode-input>
    </div>
  </div>

  <div class="rowfeed-feed">
    <div fxLayout="row" fxLayoutAlign="start center" fxLayoutGap="5px">
      <div class="material-icons">message</div>
      <div class="bold">Feed</div>
    </div>

    <div class="content-field">
      <mat-form-field style="width: 100%">
        <textarea #messageInput matInput cdkTextareaAutosize
placeholder="Comment"></textarea>
      </mat-form-field>

      <button mat-raised-button (click)="addComment(messageInput)">Add
message</button>


      <div class="comment-wrapper"
           fxLayout="column" fxLayoutGap="10px">

        <div *ngFor="let msg of messages">
          <div [id]="msg.id" class="comment-wrapper animated bounceIn fast"
fxLayout="column">
            <div class="comment-header">{{msg.sender.email}} - {{msg.created | date:
'short'}}</div>
            <div class="comment-container" fxLayout="row" fxLayoutAlign="start
center">
              <multi-mode-input class="comment-input"
[canActivate]="isCommentOwner(msg)" [value]="msg.text" [minWidthStyle]="'100%'"
                                [textArea]="true" (valueChanged)="editComment($event,
msg)">
              </multi-mode-input>

              <div class="material-icons delete-icon" *ngIf="isCommentOwner(msg)"
(click)="deleteComment(msg)">
                delete
              </div>
            </div>

          </div>
        </div>

      </div>
    </div>

  </div>
```

```
</div>
```

```
import { Injectable } from '@angular/core';
import {HttpClient, HttpParams} from '@angular/common/http';
import {Observable} from 'rxjs';
import {Board} from '../models/Board';
import {environment} from '../../environments/environment';

@Injectable()
export class BoardService {
  BASE_URL = `${environment.api_url}/boards`;

  constructor(private http: HttpClient) {}

  getBoards(): Observable<Board[]> {
    return this.http.get<Board[]>(`${this.BASE_URL}`);
  }

  getCollaborativeBoards(): Observable<Board[]> {
    return this.http.get<Board[]>(`${this.BASE_URL}/collab`);
  }

  getBoardById(id: string): Observable<Board> {
    return this.http.get<Board>(`${this.BASE_URL}/${id}`);
  }

  addBoard(board: Board): Observable<Board> {
    return this.http.post<Board>(this.BASE_URL, board);
  }

  updateBoard(board: Board): Observable<any> {
    return this.http.put<any>(this.BASE_URL, board);
  }

  deleteBoard(id: number): Observable<any> {
    return this.http.delete<any>(`${this.BASE_URL}/${id}`);
  }

}

import { Injectable } from '@angular/core';
import {environment} from '../../environments/environment';
import {HttpClient} from '@angular/common/http';
import {BoardColumn} from '../models/BoardColumn';
import {Observable} from 'rxjs';

@Injectable()
export class BoardColumnService {

  BASE_URL = `${environment.api_url}/boards`;

  constructor(private http: HttpClient) {}

  updateBoardColumn(boardId: number, boardColumn: BoardColumn):
Observable<BoardColumn> {
    return this.http.put<BoardColumn>(`${this.BASE_URL}/${boardId}/columns`,
boardColumn);
  }
```

```
  addBoardColumn(boardId: number, boardColumn: BoardColumn): Observable<BoardColumn>
{
    return this.http.post<BoardColumn>(`${this.BASE_URL}/${boardId}/columns`,
boardColumn);
  }

  deleteBoardColumn(boardId: number, id: number): Observable<any> {
   return this.http.delete(`${this.BASE_URL}/${boardId}/columns/${id}`);
  }

}

import { Injectable } from '@angular/core';
import {environment} from '../../environments/environment';
import {HttpClient, HttpParams} from '@angular/common/http';
import {Observable} from 'rxjs';
import {User} from '../models/User';

@Injectable()
export class CollaboratorService {
  BASE_URL = `${environment.api_url}/boards`;

  constructor(private http: HttpClient) {}

  inviteCollaborator(boardId: number, collabEmail: string): Observable<any> {
    let params = new HttpParams().set('collabEmail', collabEmail);

    return this.http.post(`${this.BASE_URL}/${boardId}/collab`, null, {params:
params});
  }

  deleteCollaborator(collabId: number, boardId: number): Observable<any> {
    return this.http.delete<any>(`${this.BASE_URL}/${boardId}/collab/${collabId}`);
  }

  getOwnerByBoardId(boardId: number): Observable<User> {
    return this.http.get<User>(`${this.BASE_URL}/${boardId}/collab/owner`);
  }

  getCollaboratorsByBoardId(boardId: number): Observable<User[]> {
    return this.http.get<User[]>(`${this.BASE_URL}/${boardId}/collab`);
  }

}

import { Injectable } from '@angular/core';
import {MatSnackBar} from '@angular/material';
import {HttpErrorResponse} from '@angular/common/http';
import {FormGroup} from '@angular/forms';
import {ErrorResponseDto} from '../models/dtos/ErrorResponseDto';

@Injectable()
export class ResponseService {

  constructor(private snackbar: MatSnackBar) { }

  handleMessage(message: string) {
    this.snackbar.open(message, 'OK', {
      duration: 4000
    });
  }
```

```
  handleFieldErrors(errorResponse: HttpErrorResponse, formGroup: FormGroup) {
    if (errorResponse.status === 400) {
      const error: ErrorResponseDto = errorResponse.error;
      error.fieldErrors.forEach(fieldError => {

        const formControl = formGroup.get(fieldError.field);
        if (formControl) {
          formControl.setErrors({
            serverError: fieldError.message
          });
        }

      });
    } else {
      this.handleMessage('Error occurred');
    }
  }

}

import { Injectable } from '@angular/core';
import {environment} from '../../environments/environment';
import {HttpClient} from '@angular/common/http';
import {Observable} from 'rxjs';
import {Row} from '../models/Row';

@Injectable()
export class RowService {

  BASE_URL = `${environment.api_url}/boards`;

  constructor(private http: HttpClient) {}

  updateRow(boardId: number, colId: number, row: Row): Observable<Row> {
    return this.http.put<Row>(`${this.BASE_URL}/${boardId}/columns/${colId}/rows`,
row);
  }

  addRow(boardId: number, colId: number, row: Row): Observable<Row> {
    return this.http.post<Row>(`${this.BASE_URL}/${boardId}/columns/${colId}/rows`,
row);
  }

  deleteRow(boardId: number, colId: number, id: number): Observable<any> {
    return
this.http.delete(`${this.BASE_URL}/${boardId}/columns/${colId}/rows/${id}`);
  }

}

import { Injectable } from '@angular/core';
import {environment} from '../../environments/environment';
import {RowMessage} from '../models/RowMessage';
import {Observable} from 'rxjs';
import {HttpClient} from '@angular/common/http';

@Injectable()
export class RowMessageService {

  BASE_URL = `${environment.api_url}/boards`;
```

```
  constructor(private http: HttpClient) { }

  getRowMessagesByRowId(boardId: number, colId: number, rowId: number):
Observable<RowMessage[]> {
    return
this.http.get<RowMessage[]>(`${this.BASE_URL}/${boardId}/columns/${colId}/rows/${rowI
d}/messages`);
  }

  deleteRowMessage(boardId: number, colId: number, rowId: number, id: number):
Observable<any> {
    return
this.http.delete(`${this.BASE_URL}/${boardId}/columns/${colId}/rows/${rowId}/messages
/${id}`);
  }

  update(boardId: number, colId: number, rowId: number, msg: RowMessage):
Observable<any> {
    return
this.http.put(`${this.BASE_URL}/${boardId}/columns/${colId}/rows/${rowId}/messages`,
msg);
  }
}

import { Injectable } from '@angular/core';
import {environment} from '../../environments/environment';
import {HttpClient} from '@angular/common/http';


@Injectable()
export class UserService {

  private BASE_URL = `${environment.api_url}/users`;

  constructor(private http: HttpClient) { }
}

import {FormGroup, ValidatorFn} from '@angular/forms';

export function PasswordMatch(password: string, confirmPassword: string): ValidatorFn
{
  return (group: FormGroup): { [key: string]: boolean } | null  => {
    const passwordField = group.controls[password];
    const confirmField = group.controls[confirmPassword];

    if (passwordField.value !== confirmField.value) {
      const passwordError = {passwordMatch: true};

      confirmField.setErrors(passwordError);
      return passwordError;
    }

    return null;
  };
}

import { Injectable } from '@angular/core';
import {Client} from '@stomp/stompjs';
import * as SockJS from 'sockjs-client';
import {environment} from '../../environments/environment';
```

```
import {WebSocketRowMessage} from './WebSocketRowMessage';
import {TokenStorageService} from '../auth/services/token-storage.service';

@Injectable()
export class WebsocketService {

  private stompClient: Client;

  constructor(private tokenStorage: TokenStorageService) {

  }

  initWebSocketConnection(url, onMessageReceived: (message: WebSocketRowMessage) =>
any) {
    this.stompClient = new Client();
    this.stompClient.webSocketFactory = () => new
SockJS(`${environment.api_url}/webSocket`);
    this.stompClient.connectHeaders = {
      Authorization: `Bearer ${this.tokenStorage.getToken()}`
    };
    this.stompClient.reconnectDelay = 5000;

    this.stompClient.onConnect = () => this.onConnected(url, onMessageReceived);
    this.stompClient.onStompError = () => this.onError();
    this.stompClient.activate();
  }

  sendMessage(url, message) {
    this.stompClient.publish({
        destination: `/message/send/${url}`,
        body: JSON.stringify(message)
      });
  }

  deleteMessage(url, message) {
    this.stompClient.publish({
      destination: `/message/delete/${url}`,
      body: JSON.stringify(message)
    });
  }

  private onConnected(url, onMessageReceived) {
    this.stompClient.subscribe(`/message/${url}`, (message) => {
      onMessageReceived(JSON.parse(message.body));
    });
  }

  private onError() {

  }
}
```